

Interactive Screens

More about Web Panels

GeneXus 16

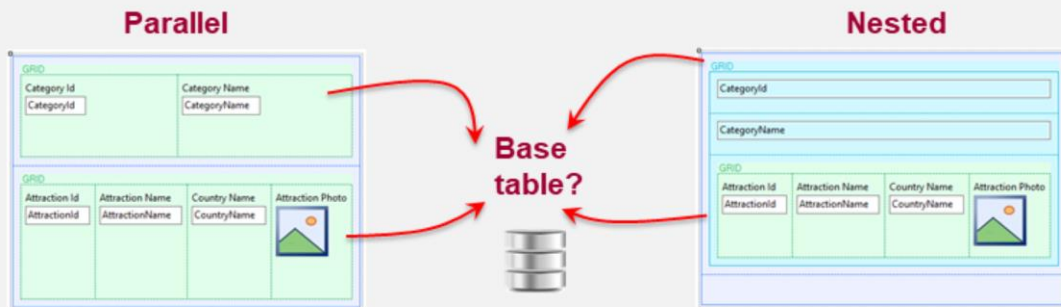
Base Tables

Previously we saw that by simply placing attributes in a grid, GeneXus understands that it must go to the database to navigate the corresponding table to retrieve the information, analogous to a For Each command.

Let's see the different cases in which GeneXus can determine a base table to be navigated.

We had seen

- Web panel without a grid or with one grid
Automatic base table?
- Web panels with multiple grids



We had seen the example of the Web Panel with "loose" attributes in the form, without a grid. Also, we had seen Web Panels that had a grid with attributes and without attributes.

And we had asked ourselves this question: when the Web Panel doesn't have any grids nor has one grid, where exactly does GeneXus search for the presence of attributes to determine whether or not to associate an implicit base table with the Web Panel? And, if attributes are found in those places, what criteria must they meet?

On the other hand, we had seen that multiple grids can be included in a Web Panel, in both parallel and nested manner. Once again, each one of those grids may or may not have an associated base table.

Web Panel without a grid



- There is no grid → There is no Base Transaction property

- Attributes in the **form** (visible or hidden)
- Attributes in **events** (outside For each commands)

Base table: minimum extended table

When the Web Panel doesn't have any grids, but it has attributes in the form (visible or hidden) or in events (as long as they are outside a For each command), the Web Panel will have a base table.

How is it determined? In the same way as a For each command to which we don't indicate a base transaction: by selecting the minimum extended table that contains all the attributes mentioned where we indicated.

The attribute(s) indicated in the Parm rule, or in the general Conditions (that is to say, in the Conditions tab), will not be taken into account when determining the base table. They will be used as filters AFTER the base table has been determined.

Web Panel with one grid



- There is one grid → base table of the web panel = base table of the grid

- **Base Transaction** property
- Attributes in the **form (grid and outside the grid, visible or hidden)**
- Grid **Order**
- Grid **Conditions**
- Grid **Data Selector**
- Attributes in **events** (outside all For each commands)

Base table: table associated with the **Base Transaction** if it has been defined; the **minimum extended table** otherwise

When the Web Panel has one grid, to determine the base table GeneXus looks at what is indicated above.

If the grid has a Base Transaction configured, its base table will be the base table of the grid/web panel. The attributes mentioned in the places indicated will have to belong to its extended table (just like in a For each command).

If the grid doesn't have a Base Transaction configured, the base table determined is the one corresponding to the minimum extended table that contains all the attributes of the places mentioned.

The attributes of the general Conditions (those of the Conditions tab) and those of the Parm rule ARE NOT included. The attributes inside a For each command are not included either.

Note that the grid can be applied a Data Selector to filter and order its data, just like we did in a For each command (and also in a Data Provider Group).

Web Panel with several grids



- There are two parallel grids → each grid may (or may not) have a base table
- There will be Refresh and Load events for each grid. There won't be a generic Load event.



Base Table of each grid:

- **Base Transaction** property
- Attributes in the **grid** (visible or hidden)
- **Grid Order**
- **Grid Conditions**
- **Grid Data Selector**
- Attributes in the grid's **Load event** (outside all For each commands)



When there is more than one grid in a Web Panel, the base table of each grid is determined by **considering only the attributes mentioned above**.

Each grid will have its Load event; the syntax is displayed above. The generic Load event cannot be used because we wouldn't know the grid it belongs to.

Unlike the case in which there was a single grid, the attributes outside For each commands in any event other than "their" Load event will not participate in determining the base tables. However, they must belong to the extended table of any of the grids. Otherwise, GeneXus will warn about it in the navigation list.

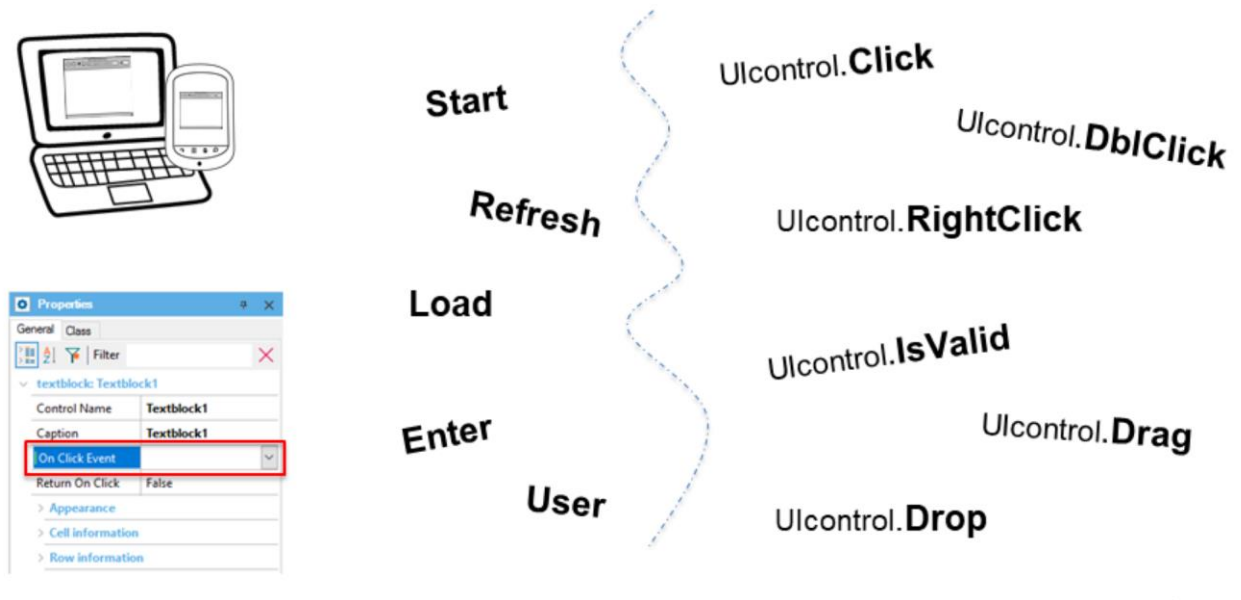
If there are attributes in the fixed part, the base table of the first grid in the form is determined by taking loose attributes into account; for the other grids they will not be taken into account.

To learn more about this special case, visit our wiki:

<http://wiki.genexus.com/commwiki/servlet/wiki?6105,Determining+the+Base+Table+for+Each+Grid+in+a+Web+Panel>

Events

Events: which ones? when? where? in what order?

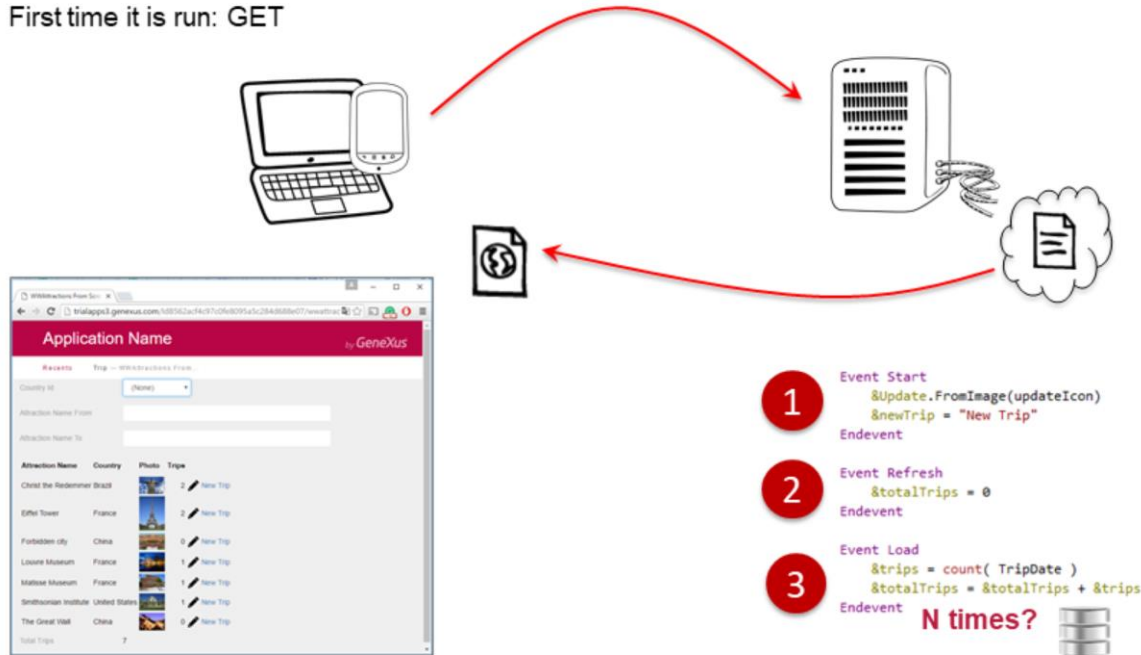


We have already seen and programmed various events in Web Panels (for example, the click event associated with controls included in the form. Depending on the control, other events such as double click, right-click, and so on can be programmed as well).

We have also introduced and used the Start, Refresh and Load events associated with the Web Panel object. The Enter event is a system event that can be associated with any control inserted in the form. It can also be run when the user presses the Enter key. We have also defined user events for buttons in an explicit manner. That is to say, we give any name to an event and associate it with a button or any other control (note the **On Click Event** property that simple controls have in the form).

Next, we will look at Web Panel events in detail, including where they are executed (in the server where the application is installed or in the client -browser) and in what order.

First time it is run: GET



In every web application we will have a machine –PC, notebook, or smart device– connected to the Internet that will allow the user to access the application through a browser; on the other hand, there will be a server containing all the application programs generated by GeneXus. Among them will be those corresponding to the Web Panels (for example, the program generated for the Web Panel that we had implemented from scratch).

What happens when we invoke the Web Panel from the browser **for the first time**? This is known as performing a GET.

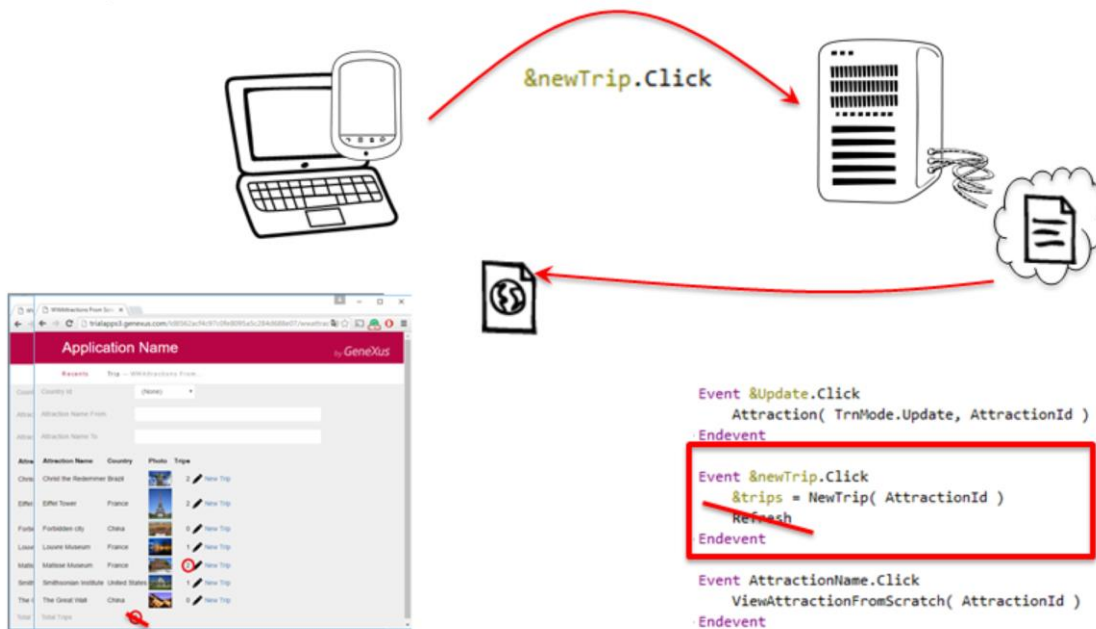
The client asks the server to run the program associated with the Web Panel and return an HTML file that indicates the browser how to draw the screen (with what data, in what format, etc.).

But, what does the program run in the server to build this HTML file?

The Start, Refresh and Load events, in that order. If it is a Web Panel with a base table, as in this case, the Load event will be run N times: one for each record that meets the conditions.

It is important to understand that the first time the Web Panel is run (GET), the user doesn't have the time to choose a value from the Dynamic Combo Box, or to filter by attraction name in the variables. It is the first call to the object. So, the `&CountryId` variable will be empty... and the condition defined indicates that the filter must be applied only if the `&CountryId` variable is not empty (when not `&CountryId.IsEmpty()`). The same will happen with the variables `&AttractionNameFrom` and `&AttractionNameTo`. Therefore, no filter will be applied and the page will be shown containing the grid loaded with all the attractions of all countries.

Subsequent executions: POST



In general, a POST takes place every time an action is performed in the client which requires going back to the server to run it.

Actions of this type may be pressing the Enter key or a button or control associated with an event.

When a POST is run, the following happens:

The information on the screen is read.

The user event that triggered the POST is executed.

For example, if the user clicks on the New Trip grid option, the AttractionId of the line on which the click was made is sent to the server. In the server, the NewTrip procedure is invoked by sending it that value of AttractionId as a parameter. Since the value returned by the procedure is loaded in the grid variable, &trips, this causes the line to be automatically loaded again in the resulting HTML file.

If we need to load everything again (for example, so that the &totalTrips variable is shown with the corresponding value), placing the Refresh command in the user event will cause the Refresh and Load events are triggered again, as we had seen in previous classes. This is run in the Server, before building the HTML file returned to the client.

Web panels with several grids: GET

- 1 Event Start
&Attractions = "Attractions"
Endevent
- 2 Event Refresh
Endevent

The screenshot shows a web form titled 'CategoryAndAttractions' with two grids. The left grid, 'GRID', has columns: Category Id (CategoryId), Category Name (CategoryName), and &Attractions. The right grid, 'GRID', has columns: Attraction Id (AttractionId), Attraction Name (AttractionName), Country Name (CountryName), and Attraction Photo. Events are assigned to each grid: Grid1.Refresh (3), Grid1.Load (4), Grid2.Refresh (5), and Grid2.Load (6). The Load events are annotated with 'N times if BT' and '1 otherwise'. A red arrow labeled 'F5' points to the right.

For Web Panels with several grids, the Refresh event that will now be generic (for the entire Web Panel) will launch the Refresh and Load events of every grid.

Depending on whether the grid has a base table or not, the Load event, just like for the case of a single grid, will be run N times, once for every record in the base table to be loaded as a grid line; otherwise, if it doesn't have a base table, it will be launched only once.

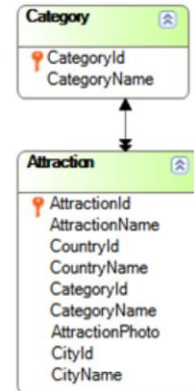
The triggering order of events is displayed above. It will depend on the order in which the grids are placed on the screen (from left to right, from top to bottom).

Web panels with several grids: GET

The screenshot shows a web application interface with a header 'Application Name by GeneXus'. Below the header, there are two tabs: 'Recents' and 'Category And Attra...'. The main content area contains two grids. The first grid, 'Category Name', has columns for 'Category Name', 'Attractions', 'Attraction Name', 'Country Name', and 'Attraction Photo'. The 'Attractions' column is highlighted with a red box. The second grid, 'Attraction', has columns for 'Attraction Name', 'Country Name', and 'Attraction Photo'. A red arrow points from the 'Attractions' column of the first grid to the second grid, with the text 'Load the attractions of this category' next to it.

Category Name	Attractions	Attraction Name	Country Name	Attraction Photo
Museum	Attractions	Louvre Museum	France	
Monument	Attractions	The Great Wall	China	
Tourist Site	Attractions	Eiffel Tower	France	
Square	Attractions	Christ the Redemmer	Brazil	
Museum	Attractions	Smithsonian Institute	United States	
Monument	Attractions	Matisse Museum	France	
Tourist Site	Attractions	Forbidden city	China	
Square	Attractions			

Even though the base tables of every grid have a 1 to N relationship, the loadings are not automatically related.



For the attractions grid to display only those of the category selected in the grid to the left, we will have to indicate the filter explicitly.

To this end, we have added a column with the &Attractions variable, of character(15) type, with the Attractions text (we have indicated in this way in the Start event, because it is defined when the Web Panel is opened. It won't change later). Thus, when the user clicks on this option, we must ask for the attractions grid to be loaded (again).

Web panels with several grids: POST

Event `&Attractions.Click`
`&CategoryId = CategoryId`
 ·Endevent

When is it refreshed?

To program the attractions grid loading from the category selected by the user in the other grid, we create a `&CategoryId` variable to which we will assign a value every time the user clicks on “Attractions” for the desired line. Note that we have entered a condition for an attraction to be loaded in the grid (Grid2), which indicates that its `CategoryId` must match the variable value.

In this way, the first time it is run no attractions will meet the condition, because for this case the `&CategoryId` variable will be empty.

Later, when the user clicks on `&Attractions`, the variable is given a value based on the `CategoryId` of that line. But if we don't do anything else, Grid2 will never be refreshed. We will have to ask Grid2 to be refreshed.

Web panels with several grids: POST

```
Event &Attractions.Click  
&CategoryId = CategoryId  
Grid2.Refresh()  
Endevent
```

The screenshot displays the GeneXus IDE interface. At the top, a browser window shows the application's header with the text "Application Name" and "by GeneXus". Below the header, there is a table with columns: "Category Name", "Attraction Name", "Country Name", and "Attraction Photo". The table contains two rows of data:

Category Name	Attraction Name	Country Name	Attraction Photo
Museum	The Great Wall	China	
Monument	Forbidden city	China	

The "Attractions" link under the "Tourist Site" category is circled in red, with a red arrow pointing to the "Attraction Name" column. To the right, the "Properties" window is open, showing the "Grid: Grid2" properties. The "Conditions" property is set to "CategoryId = &CategoryId;".

We do so with the Refresh method of that grid.

Web panels with nested grids

The screenshot displays the GeneXus IDE interface for a web panel titled "CategoryAndAttractionsNested". The main workspace shows a "Web Form" with a "MainTable" and a "Grid2". A "GRID" is highlighted in blue, containing a "CategoryName" field. Below it, a "GRID" is highlighted in green, containing three columns: "Attraction Id" (with "AttractionId" below), "Attraction Name" (with "AttractionName" below), and "Attraction Photo" (with an image icon below). To the right, a data table is visible with columns "Attraction Id", "Attraction Name", and "Attraction Photo". The table contains three rows of data:

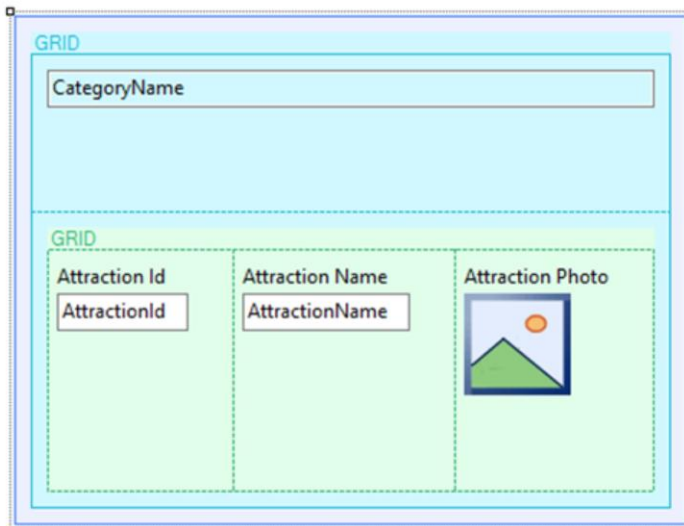
Attraction Id	Attraction Name	Attraction Photo
22	Louvre Museum	
26	Smithsonian Institute	
27	Matisse Museum	

Below the table, there are two more sections: "Monument" with rows for "24 Eiffel Tower" and "25 Christ the Redemptor", and "Tourist Site" with row "23 The Great Wall". A "Toolbox" on the right lists "Controls" and "Containers". The "Containers" section includes "Free Style Grid", "Grid", "Group", "Html", "Responsive Table", "Section", "Tab", and "Table". Red arrows point from the "GRID" labels in the workspace to the "Grid" and "Free Style Grid" items in the toolbox.

To nest grids, Freestyle grids are used. For example, instead of seeing all categories and clicking on one to view its attractions, we can see them all nested; that is to say, for each category, all its attractions. As if it was a case of nested For each commands. It is similar.

Here, navigations are related.

Web panels with nested grids (GET)

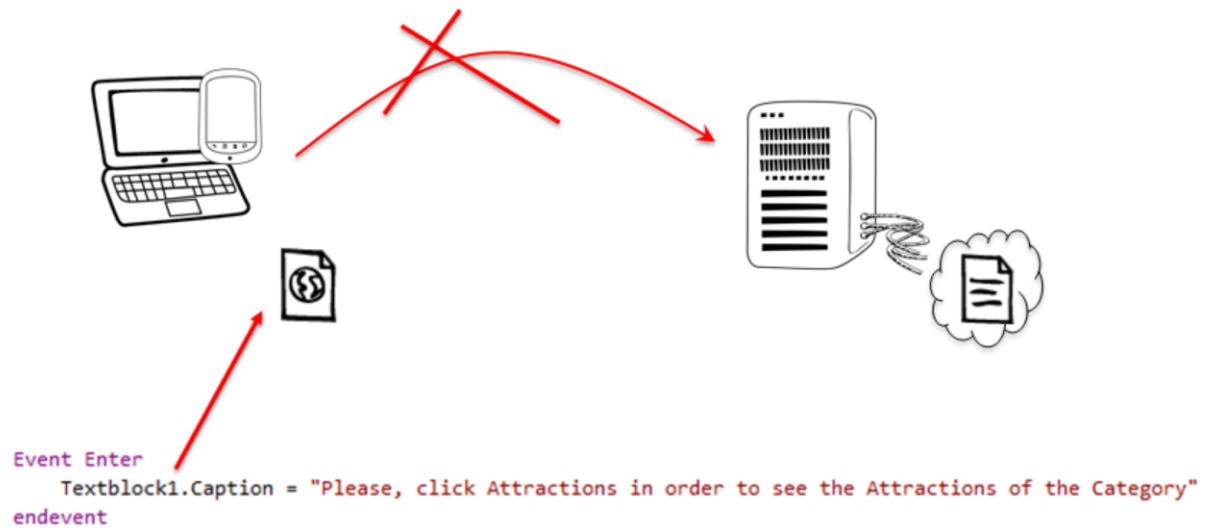


1. Start
2. Refresh
3. Grid1.Refresh
4. Grid1.Load
5. Grid2.Refresh
6. Grid2.Load

The triggering order of events is the expected one. First, the Start event of the Web Panel, then Refresh, and finally the Refresh event of the grid and Load. If it has a base table, the Load event will be triggered N times, one for each line. Otherwise, it will be triggered only once.

In any case, after the Load of the Freestyle grid, the Refresh and Load events of the nested Grid will be triggered. Again, this second Load may be run only once or N times, depending on whether there is a base table or not.

Some events can be solved in the client, without a POST to the Server



Not every action performed by the user and associated with an event will cause a POST to the server. Some of them can be solved in the client itself.

For example, if in an Enter or user event, or in an event associated with a control, a control becomes invisible, or its Caption or color is changed, it is solved in the client itself.

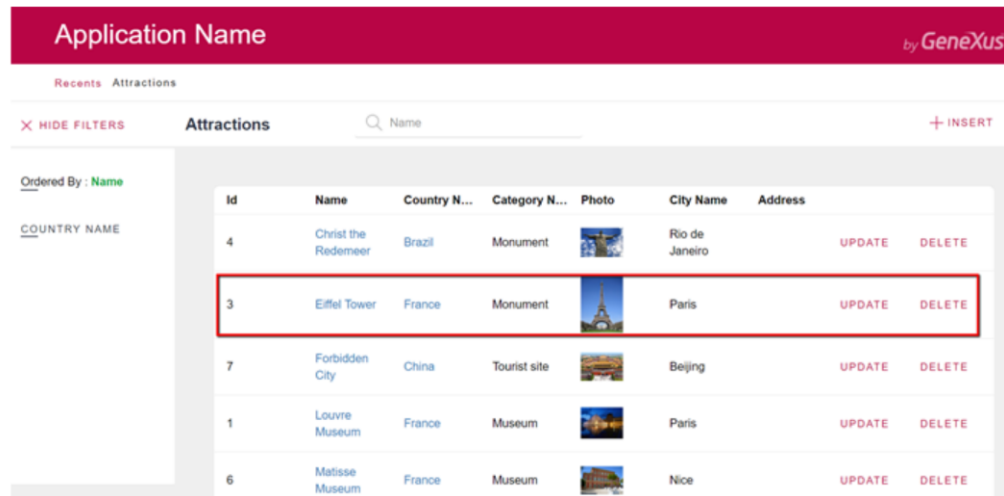
Multiple Selection in a Grid

Many times we need to work with a selected group of elements to do something with them later on.






Since a grid allows us to view many elements, it's only natural that we want to select several elements in a grid.

Next, we will see how to make a multiple selection in a grid and how to run through the elements to process them.

Selecting a single row in a grid



The screenshot shows a web application interface for managing tourist attractions. At the top, there is a header with the application name and the GeneXus logo. Below the header, there is a navigation bar with a search field and an 'INSERT' button. The main content area displays a grid of attractions. The grid has columns for Id, Name, Country Name, Category Name, Photo, City Name, and Address. The row for 'Eiffel Tower' is highlighted with a red border, indicating it is selected. The grid also includes 'UPDATE' and 'DELETE' buttons for each row.

Id	Name	Country N...	Category N...	Photo	City Name	Address
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro	UPDATE DELETE
3	Eiffel Tower	France	Monument		Paris	UPDATE DELETE
7	Forbidden City	China	Tourist site		Beijing	UPDATE DELETE
1	Louvre Museum	France	Museum		Paris	UPDATE DELETE
6	Matisse Museum	France	Museum		Nice	UPDATE DELETE

For example, suppose that we have a grid with data on tourist attractions.

To work with a specific attraction, we need to mark a grid row as selected, in order to access the values of each column in this row.

To select a single row in a grid: AllowSelection=True

Behavior








Sortable	True
Allow Drop	False
Allow Drag	False
Notify Context Change	False
Allow Collapsing	False
Allow Selection	True
Allow Hovering	True

Selected Line

Marked Line

Application Name

Recents Show Attractions

Attraction Id	Attraction Name	Attraction Photo
1	Louvre Museum	
2	The Great Wall	
3	Eiffel Tower	
4	Christ the Redemeer	
5	Smithsonian Institute	
6	Matisse Museum	
7	Forbidden City	

In a web application, if we want to mark a grid row as selected, we open the grid properties and set the AllowSelection property to True.

This value also enables the Allow Hovering property, which is True by default. This means that when the mouse passes over the rows, they are painted in one color.

When AllowSelection is set to True, clicking on an attraction causes the row to be highlighted with another color. These colors can be configured in the Theme object, in the classes assigned to the properties SelectedRow Class and Hover Row Class of the Grid class.

Selecting multiple rows in a grid



Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<-default>
Selection Type	Platform Default
Enable Multiple Selection	True
Pull To Refresh	False
Inverse Loading	False
Default Selected Item Layout	(none)
Show Selector	Always
Selection Flag	
Selection Flag Field Specifier	

Name	Type
& Variables	
& Standard Variables	
Selected	Boolean

On Web environments



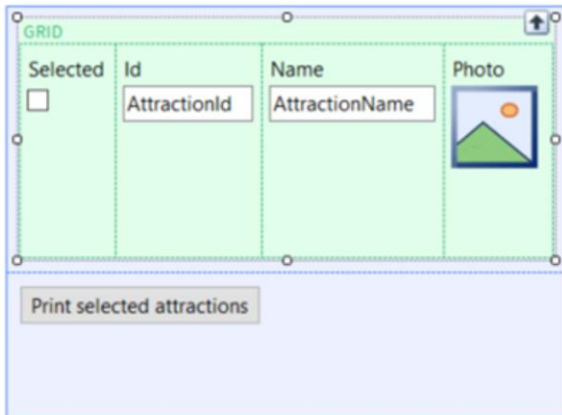
Suppose that the travel agency requires that, from a grid of attractions, we select those that we're interested in and print a list with the selected attractions.

In a Web application, we can use a grid property to select a single row, as we've seen, but there aren't any properties for us to indicate that we want to select several rows at the same time.

On the other hand, this can be done in a Smart Device application using the grid property **Enable Multiple Selection**. In order to display the checkbox that allows us to select the row, we use the Show Selector property.

Since we need a web screen to meet the travel agency's request, we must solve this differently. A solution would be to add a variable of Boolean type to the grid, use it to mark the selected row, and then run through the grid rows to process those that were selected. Let's see how to do this.

Selecting multiple rows in a grid



Name	Type	Is Collection
& Variables		
& Standard Variables		
Selected	Boolean	<input type="checkbox"/>
SelectedAttractionsIds	Id	<input checked="" type="checkbox"/>



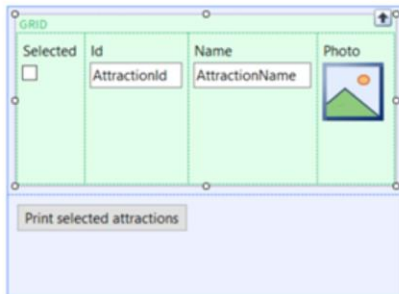
First, we will create a web panel called SelectedAttractions, to which we will add a grid with the attributes AttractionId, AttractionName and AttractionPhoto.

Next, we will create a variable called Selected, of Boolean type and add it to the grid, in the leftmost column.

We will also add a button and give the name "Print selected attractions" to the event. In this event, we will invoke the procedure object that prints the selected attractions. To send them to it, we must first save them in a collection.

To do so, we add a variable called SelectedAttractionsIds, since it is enough for us to save in the collection only the identifiers of the selected attractions. We assign it the ID data type and mark it as a collection.

Selecting multiple rows in a grid



Name	Type	Is Collection
Variables		
Standard Variables		
JsonSelectedAttractions	LongVarChar(2M)	<input type="checkbox"/>
Selected	Boolean	<input type="checkbox"/>
SelectedAttractionsIds	Id	<input checked="" type="checkbox"/>

```

1 | Event 'Print selected attractions'
2 |   For each line in GridAttractions
3 |     if &Selected
4 |       &SelectedAttractionsIds.Add(AttractionId)
5 |     endif
6 |   endfor
7 |   if &SelectedAttractionsIds.Count > 0
8 |     &JsonSelectedAttractions = &SelectedAttractionsIds.ToJson()
9 |   endif
10 |   SelectedAttractions(&JsonSelectedAttractions)
11 | Endevent

```



To run through the rows of a grid, we use the instruction **For each line** ... in ... the name of the grid, which in this case is GridAttractions. This instruction will be positioned in each grid line and will allow us to retrieve the values taken by its columns in each row. One important thing to keep in mind is that the For each line command will only run through the records loaded in the grid. If there are more records on the following pages, they will not be included. Only those records displayed in the grid lines can be run through.

To find out if the attraction was selected or not, we type If &Selected. If this is the case, we add AttractionId to the collection, so we type &SelectedAttractionsId.Add(AttractionId), and close the If clause. Now we close the For each line clause with endfor. If we were using a Smart Device application, instead of the Boolean variable we would use the **Show Selector** property of the grid. To run through the rows we use the command **For each selected line**.

Going back to our web panel, once all the grid rows have been run through, we will send the collection of selected attractions to the list. But in order to do so, we can't send the collection variable as a parameter. Instead, we have to serialize its content; that is, generate a text file in a structured format, such as JSON or XML.

We will create a variable to save the JSON, and call it JJsonSelectedAttractions of LongVarChar type.... then we load it from the collection, using the ToJson method.

We now invoke the procedure SelectedAttractions and send it via parameter the JSON we previously obtained.

Selecting multiple rows in a grid

```

1 | print Titles
2 |
3 | &SelectedAttractionsIds.FromJson(&JsonSelectedAttractions)
4 |
5 | For &AttractionId in &SelectedAttractionsIds
6 |   For each Attraction
7 |     Where AttractionId = &AttractionId
8 |     &AttractionName = AttractionName
9 |     &AttractionPhoto = AttractionPhoto
10 |   Endfor
11 |   print Attractions
12 | Endfor

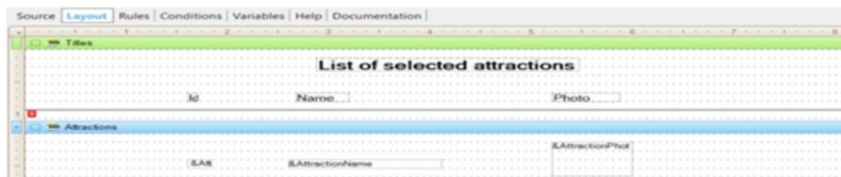
```

Name	Type
& Variables	
& Standard Variables	
& Autodefined Variables	
AttractionId	Attribute:AttractionId
AttractionName	Attribute:AttractionName
AttractionPhoto	Attribute:AttractionPhoto
JsonSelectedAttractions	LongVarChar(2M)
SelectedAttractionsIds	Id

```

1 | Parm(in:&JsonSelectedAttractions);
2 | Output_file('SelectedAttractions', 'PDF');

```



We create an object of procedure type and call it SelectedAttractions. Also, we define the variables AttractionId, AttractionName and AttractionPhoto. Then we create a variable called JJsonSelectedAttractions, of LongVarChar type and a variable SelectedAttractionsIds of ID type, which we mark as a collection.

Now we open the rules and type a Parm rule that receives the variable JJsonSelectedAttractions as input. Now we include the rule Output_file... and mark the procedure as Main and the Call Protocol as HTTP.

In Layout, we rename the printblock to Titles and drag a Text Block to which we add the text "List of selected attractions." We also add labels for the columns of the list: "Id", "Name" and "Photo." Finally, we add a line under these column labels.

Now we insert another printblock, call it Attractions, and drag the variables AttractionId, AttractionName and AttractionPhoto.

In the source we type print Titles.... And load the collection variable SelectedAttractionsId with the content of the variable JJsonSelectedAttractions, using the FromJson method.

In this way, the IDs of the selected attractions were saved in the collection; we need to run through them to access them, so we type: For &AttractionId in &SelectedAttractionsIds, and then with each AttractionId accessed we retrieve the name of the attraction and its photo. To do so, we type For each Attraction, Where AttractionId = &AttractionId and load the variables &AttractionName and &AttractionPhoto from the corresponding attributes.





We close the For Each command, print the printblock with the attractions' information and finally close the For in clause.

Now we can test what we've programmed, so we press F5.

Selecting multiple rows in a grid



Application Name


Recents Select Attractions

Selected	Id	Name	Photo
<input checked="" type="checkbox"/>	1	Louvre Museum	
<input type="checkbox"/>	2	Great Wall	
<input type="checkbox"/>	3	Eiffel Tower	
<input checked="" type="checkbox"/>	4	Christ Redeemer	

Print selected attractions

List of selected attractions

Id	Name	Photo
1	Louvre Museum	
4	Christ Redeemer	



In the Developer Menu, we click on SelectAttractions and see that the web panel is opened, showing the list of attractions. Note that the control associated with the Boolean variable Selected is a check box that allows selecting attractions.

We choose the Louvre Museum and the Eiffel Tower... and press Print selected attractions. We see that the list opens, showing the attractions we had selected.

In summary...

- To work with a grid row, the property AllowSelection is used.
- The grid of an SD object has the property Enable Multiple Selection but the grid of a Web object doesn't.
- To run through the rows of a grid, For each line is used in a Web environment, and For each selected line is used in an SD app.
- Note: It is not possible to send a collection via parameter to a PDF report; it must be converted to JSON/XML to send the string.

Let's review the concepts seen so far.

To select a single row in a grid, we set the AllowSelection property to True in order to access the values of each column of the grid, for the selected row.

To select more than one row at the same time, in a Smart Device application the grid has the property Enable MultipleSelection. By setting it to True, you can select multiple rows for further processing. In a web application grid we don't have this property, so we need to use a Boolean variable in the grid that will allow us to save the row selection.

To run through the rows of a grid in a web application we use the For each line command. This command will iterate only for the rows that are loaded in the grid at a given moment, and in each iteration we will have the values of the grid columns for the row run through. To know if a row was selected, we use the value of a Boolean variable that we added for that purpose.

In an SD application we use the property Show Selector and the command For each selected line.

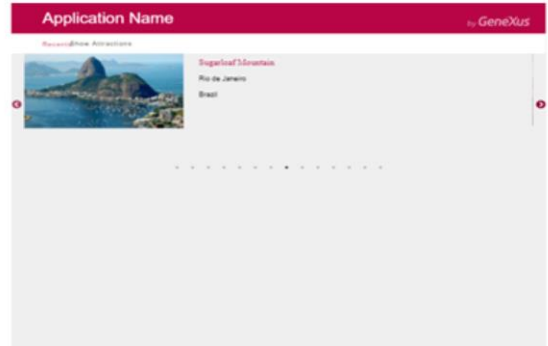
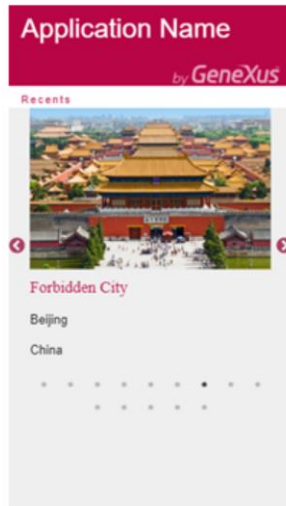
It is not possible to send a collection variable via parameter to a web object, as a web panel or a procedure with HTTP protocol. To send the details of the selected attractions we use a collection, but then serialize that information generating a structured text file; for example, in JSON or XML format.

Changing the behavior of a grid

By changing the value of some grid properties we can make it change the way information is displayed; for example, show data horizontally, with infinite scroll, or flexibly, customizing the display according to its content.

Let's look at some cases.

Horizontal Grid



A horizontal grid is a grid that allows us to show the information as if it were a carousel, that is to say, we can apply paging from the right and the elements will move horizontally, to the left.

Horizontal Grid

Grid: Grid1	
Control Name	Grid1
Collection	
Base Trn	
Order	
Conditions	
Unique	
Data Selector	(none)
Appearance	
Class	Grid
Custom Render	HorizontalGrid
Empty Grid Text	
Auto Resize	True

HorizontalGrid	
Paged	True
Show Page Controller	True
Page Controller Class	GridPageController
Show Arrows	True
Infinite	False
Auto Play	False
Variable Width	False
> Rows per page	1
Layout	
Cell Padding	1
Cell Spacing	2

To make a grid display its content horizontally, i.e. in carrousel format, we need to set the Custom Render property to HorizontalGrid.

By assigning this value, a series of properties will be enabled through which we will be able to configure how the grid will look like. For example, the Show Page Controller property determines if the pager will be displayed or not; the Page Controller Class property will allow us to define the **class** on which this pager is based.

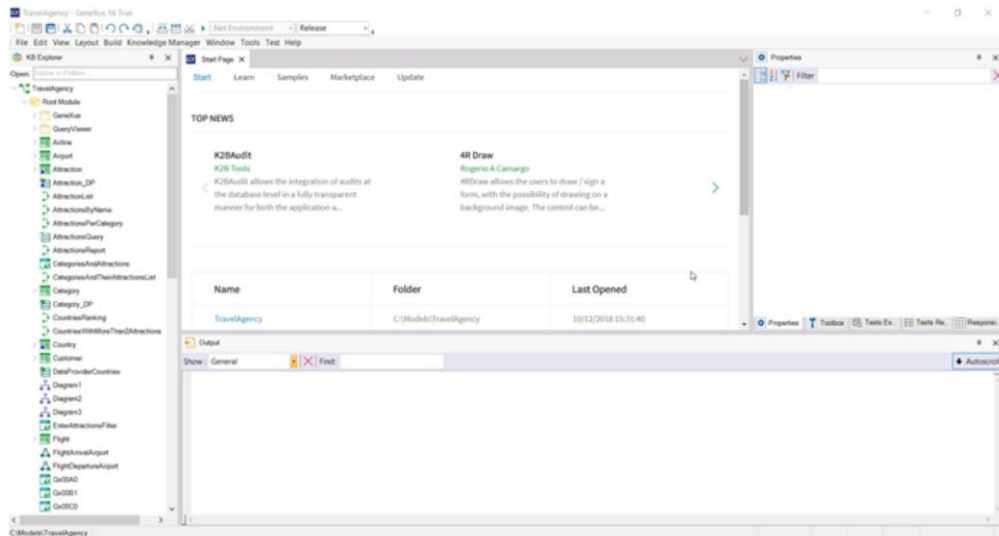
This class is like a template to which we can change properties and all controls based on the class will inherit these values.

There is a certain parallelism with the concept of domain that we saw before. By changing the values of the properties of a class, we will be able to customize all the controls that are based on it.

These classes, which define the appearance and behavior of each control on the screen, are grouped below the **Theme object**, which we will see later.

Returning to the properties that allow us to customize our horizontal grid, we see that we can decide whether we want to see the grid's arrows that we use to change the record, define if the grid's width can be changed, how it will be paged, etc.

Let's see an example of a grid of this type.



[Demo: <https://youtu.be/b76ee3lu3hQ>]

Let's see a demo of how to configure a horizontal grid using the **Custom Render** property. To try it, we can use a web panel called ShowAttractions with a grid. More specifically, it is a FreeStyleGrid, but the example is also valid for a standard grid. To the grid we add the attributes AttractionName, AttractionPhoto, CityName and CountryName, as well as a responsive table to improve alignment.

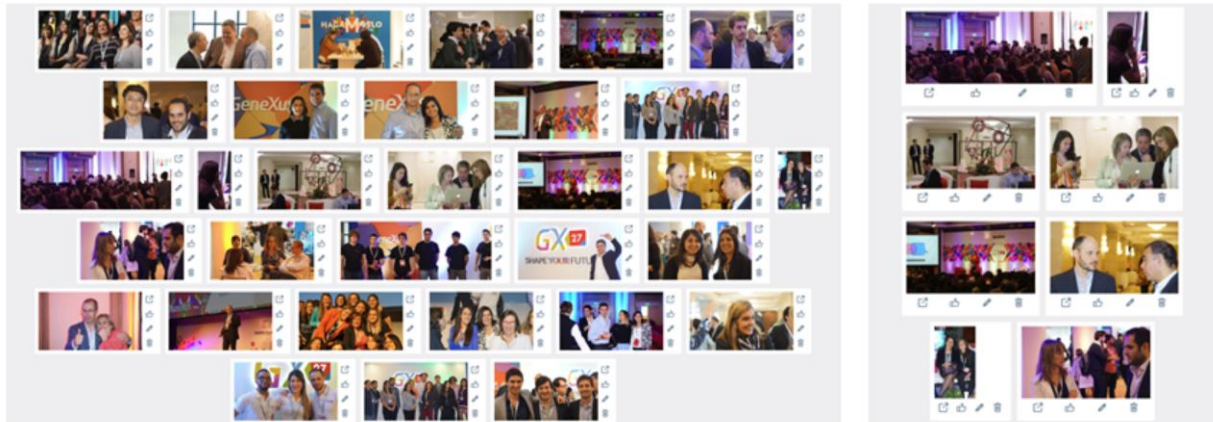
If we run this web panel... we can see the grid with its usual behavior. Now we set the **Custom Render** property to Horizontal grid and run.

Note that just by doing this, the information is displayed horizontally, as in a carrousel, showing one attraction at a time.

We also see the paging control of the carrousel and the arrows to the right and left of the page. As we said, all this is can be configured from the Theme object.

Since we're using Chrome, by pressing F12 we can select the screen size. By selecting other devices we can see that the information is adjusted in a responsive manner, even if we change its orientation.

Flexible Grid



A flexible grid allows us to view information in a way that can be adapted to our needs, and allows for greater content customization in the visualization of the content, than allowed by the responsive behavior.

When we add photos, for example, we never know the length or width of each element to be displayed. These elements should be automatically rearranged as best as possible, depending on the screen size. For example, in the figure on the left the photos show actions on the right, and in the figure on the right, in a smaller screen, the elements were adjusted to the available width and the controls of the photos were automatically placed below them.

Flexible Grid

Grid: Grid1

Control Name	Grid1
Collection	
Base Trn	
Order	
Conditions	
Unique	
Data Selector	(none)

Appearance

Class	Grid
Custom Render	FlexGrid
Empty Grid Text	
Auto Resize	True

FlexGrid

Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Align Content	Stretch

To make a grid display its content in a flexible manner, we must set the **Custom Render** property to FlexGrid.

By assigning this value, a series of properties will be enabled through which we will be able to configure how the grid will look like.

Flexible Grid

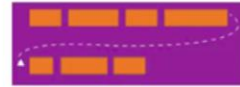
FlexGrid

Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Align Content	Stretch

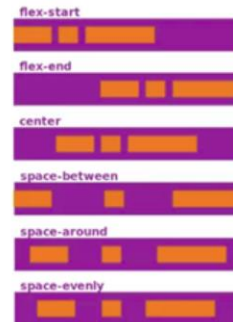
Flex Direction



Flex Wrap



Justify Content



The **Flex Direction** property allows you to change the direction in which items are displayed on the screen, whether it is by row or inverted row, by column or inverted column.

Flex Wrap sets how items will be arranged when they take more than the length of a row and not all of them can be displayed. The Wrap value arranges them in the following line.

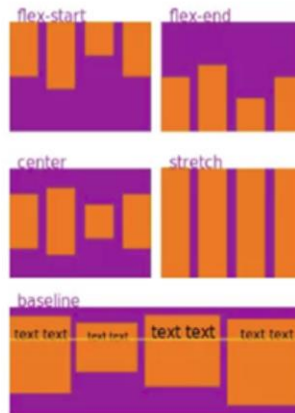
Justify Content allows us to set how content will be aligned. It can be from left to right, from right to left, centered; also, the gap can be adjusted to take the entire available width, or spaces can be added between elements.

Flexible Grid

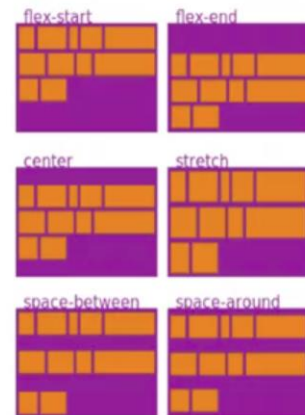
FlexGrid

Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Align Content	Stretch

Align Items



Align Content



The **Align Items** property allows us to select how the items are aligned in the rows.

Align Content allows us to align the row when there is extra space in the container.

Grid with infinite scroll

Application Name

by GeneXus

Recents Attractions

X HIDE FILTERS





Attractions

Q Name

+ INSERT

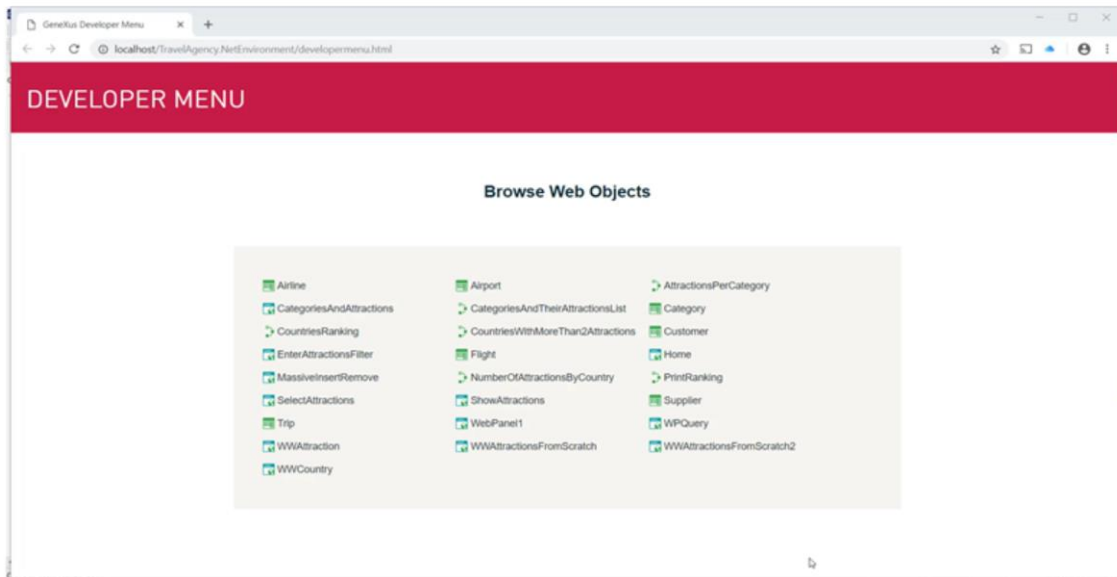
Ordered By : Name

COUNTRY NAME

Id	Name	Country Name	Category Name	Photo	City Name	Address		
10	Copacabana Beach	Brazil	Tourist site		Rio de Janeiro		UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris		UPDATE	DELETE
7	Forbidden City	China	Tourist site		Beijing		UPDATE	DELETE
1	Louvre Museum	France	Museum		Paris		UPDATE	DELETE

Loading...

Instead of showing the grid contents with paging, we may want to load the lines as we scroll, on the same page.



[Demo: <https://youtu.be/jWNAkRXwGw0>]

For this demo, we've added some attractions. If we open "work with Attractions", we can see 10 attractions displayed on the page, and at the bottom of the page there is a paging control. Pressing on the Next button shows 4 more attractions.

To change the default value from 10 to 5 records per page, we open the WorkWithAttraction pattern instance, click on the Selection node, in the property **Rows per page** we select **<custom>**, and in the property **Custom Rows** we select the value 5.

We press F5....And see that the attractions are separated in 3 pages.

If we needed to review information of the attractions and instead of 14 attractions we had 1000, even leaving the default value of 10 records per page, there would be many pages to review. It would be much easier if we could make the grid load the records automatically, as we scroll down, without having to manually apply paging. This is known as ***infinite scroll***.

To obtain this behavior in the WorkWith pattern grid, we open the Selection node and in the **Paging mode** property we select the value **<infinite scrolling>**. We press F5 and see that the scroll bar is displayed in the grid, and that the paging control is hidden. Moving the scroll bar shows the message "Loading..." to indicate that the database records are being loaded. We can scroll indefinitely until showing all the attractions loaded in the table.

If we want to change the number of records per page in a standard grid not included in a Work With pattern, we need to change the default value of the **Rows** property, which is 0, for the desired value. The value 0 equals to "unlimited" and makes all the grid records visible at once.

We press F5... And see that the paging control disappears and the scroll bar does not appear in the grid. The scroll appears on the page, because the grid is too long to be entirely displayed, and the page makes us scroll. (Note: To view the image in a larger size than the default one, edit the column AttractionPhoto, set its Auto Resize property to False and change the values of the properties Height and Width).

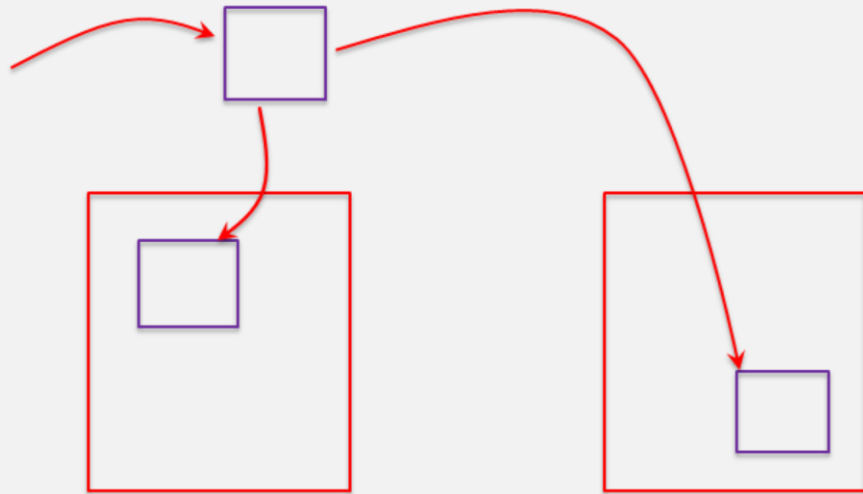
To obtain infinite scroll in a standard grid not included in a Work With pattern, we set the **Rows** property to a value other than 0 and change the **Paging** property to **Infinite Scrolling**.

In a Free Style grid, the default value of the Rows property is <unlimited> and this value cannot be changed.

Web Components

We had seen

- Types of Web Panels:
 - Web page
 - Component
 - Master Page



In a previous class we had seen that there are three types of Web Panels.

In these videos we have only seen the Web Page type, but there are Web Panels that can be defined as components, for the case when part of a web page has to be repeated in multiple web panels. When it is defined as a component, it can be inserted in other objects with a web screen.

Web components

If we repeat it in several panels

Web Component

Properties	
Filter	
Web Component: CategoryAttracti...	
Name	CategoryAttractions
Description	Category Attractions
Module/Folder	Root Module
Theme	Carmine
Type	Component
URL access	No
Show Master Pa	False
Main program	False

For example, suppose that the grid showing the attractions of the selected category is repeated in several Web Panels.

So, it will be better to have a Web Panel of Component type with it and its entire programming (for example, we have replaced the condition we had at the grid level, with the Parm rule that receives a value in the CategoryId attribute that, as we know, applies an automatic filter).

Then...

Web components

The screenshot shows the GeneXus IDE interface. At the top, there's a menu bar with 'Web Form', 'Rules', 'Events', 'Conditions', and 'Variables'. Below that, a toolbar shows 'MainTable' and 'Component1'. The main workspace contains a 'Textblock1' and a 'GRID'. The grid has three columns: 'Category Id' (with a sub-column 'CategoryId'), 'Category Name' (with a sub-column 'CategoryName'), and '&Attractions'. A red arrow points from the '&Attractions' column to a '<Component>' control. Another red arrow points from the '<Component>' control to the 'Properties' window. The 'Properties' window shows the 'Object' property set to 'CategoryAttractions' and the 'Parameters' property set to '&CategoryId'.

```

Event &Attractions.Click
  &CategoryId = CategoryId
  Component1.Object = Create( CategoryAttractions, &CategoryId )
EndEvent

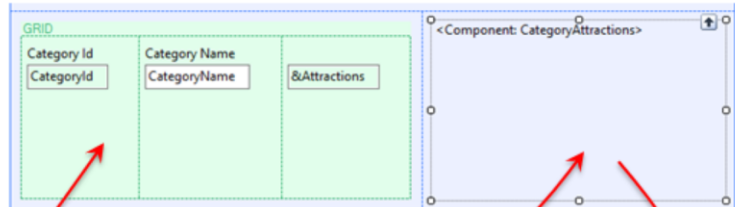
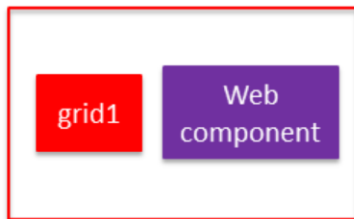
```

In the Web Panel where we initially had the attraction grid, we replace it with a Web Component control. Now we only need to indicate what object of Web Component type will be loaded there. We can do it through programming or through the Component control properties, as we see above. There we indicate the object that will be loaded, and the parameter we have to send it. In this case, it is the value of the &CategoryId variable.

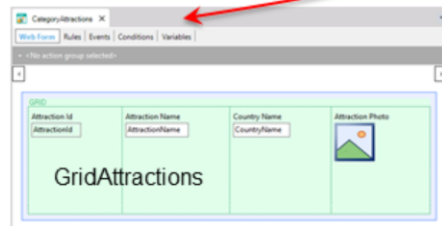
Now we also have to change the programming of the click event of the categories grid. Once we have given the &CategoryId variable the value of the line category, we need to have the Web Component created again inside the control, so that it can receive the variable value in a parameter. To this end, we will use the Object property at the programming level.

The Refresh() method exists at the component level. We will see it later after talking about the events run in the GET.

Events (GET)



1. Start
2. Refresh
3. Grid1.Refresh
4. Grid1.Load
5. Start
6. Refresh
7. Refresh GridAttractions
8. Load GridAttractions



If there is a Web Component inside a Web Panel, the events triggered and their order are the expected ones. The following is triggered:

1. Start Event of the web panel
2. General Refresh event of the web panel
3. Refresh event of grid1
4. Load event of grid1
5. Component events (Start, Refresh, Refresh and Load of every grid found)

Next, if there is a user or control event at the component level, this event is run and only the corresponding part of the component is refreshed. Here you will find more information: <http://wiki.genexus.com/commwiki/servlet/wiki?22472,Event+Execution+Scheme>,

Web components

```

Event &Attractions.Click
  &CategoryId = CategoryId
  Component1.Object = Create( CategoryAttractions, &CategoryId )
Endevent

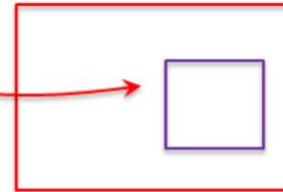
```

```

Event &Attractions.Click
  &CategoryId = CategoryId
  component1.Refresh()
Endevent

```

Web Component: Component1	
Control Name	Component1
Object	CategoryAttractions ...
Parameters	&CategoryId
Cell information Row information	



1st Create
(&CategoryId empty)

Trigger:

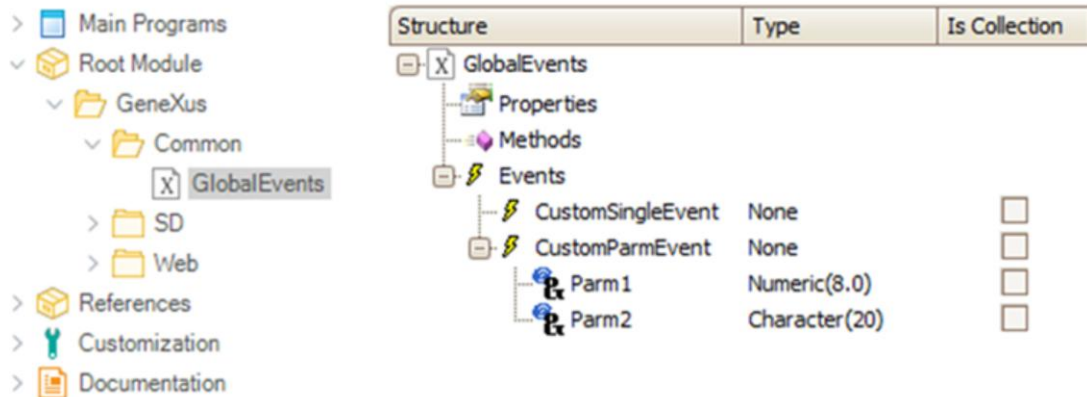
- Start
- Refresh
- Load

The Refresh method at the Web Component level will trigger the Start, Refresh and Load events of the web component, and will load its entire screen area again. Why haven't we used it? Because of the parameter that we want to send the object that we created there. The Refresh method will send, through a parameter, the value that &CategoryId had when the Create operation was run (in this case it will be when the GET was performed; that is to say, when the screen was drawn for the first time). At that moment, the variable was empty.

We will not study this in detail here.

Global Events

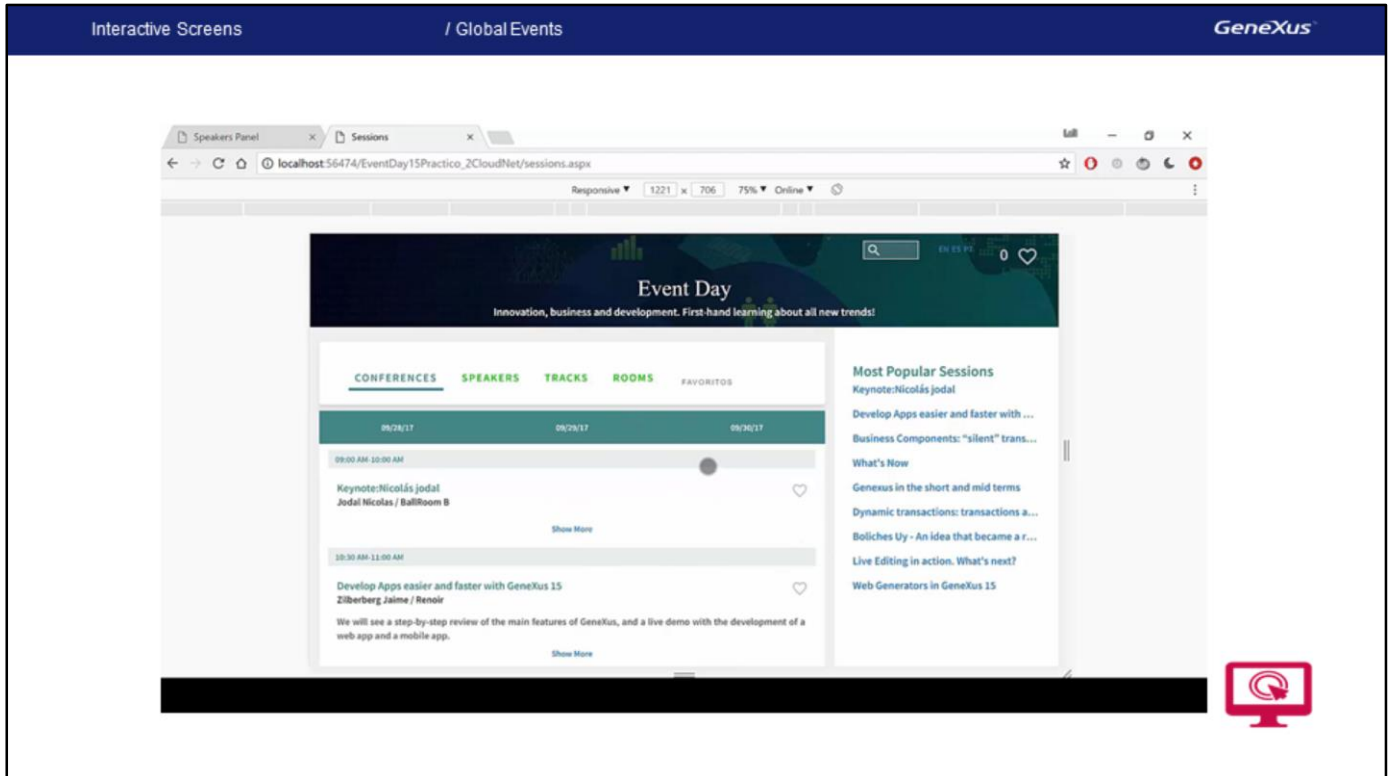
Global Events



Sometimes we want to develop an application where an action in a component causes a reaction in a different component of the screen

For example – we want to trigger an action in a web component when the user selects a menu element that is located in a different component.

Since communication between web components is limited - the only possible communication is from parent to children - global events allow us to establish communication between web components that are not related.



[DEMO: <https://youtu.be/D8akim-gRSQ>]

Suppose there's a web application for an event. If we go to the component that displays the sessions, clicking on the Favorite button will refresh a component that shows all favorites, which is on the master page and has no direct relationship with the component that is triggering it. Let's see how to do this in GeneXus.

To achieve this, there is a new object called GlobalEvents. We can create as many Global Events objects as we want; basically, it is an external object and we could create GlobalEventsBackend, GlobalEventfrontend, etc. depending on how we want to group them. Remember that they are global and therefore it is important to use them with certain criteria and in an orderly manner so that they are not too intrusive.

Our GlobalEvent here mainly updates the number of favorites.

In this case, our Event does not receive parameters, but it is possible to implement events that do; if it were necessary to send parameters they would continue defining them, associated with the event, with their data types, etc. Then, when making the invocation, the object that triggers the event is responsible for defining the parameters, and all the components that implement the event or that listen to this event will receive the parameters with this format.

How is this global event triggered? In this case, the Sessions component is used. When the favorites image is clicked on, the toggle that shows the image in gray or yellow is enabled, and the GlobalEvent is triggered.

Who receives/runs this GlobalEvent? That's our decision, and it can be done by any component that can reasonably do it.

In our case, the component that shows the number of favorite events is doing it; basically, this component implements GlobalEvents.UpdateFavoritesQuantity and here it will make the call needed to update the information, which could be a refresh, etc.

In short, if we had more than one component that needed to subscribe to this event, it is possible to do so.

For example, we might want to show favorites' information in different parts of the screens, and when someone adds an event or session to their favorites, we could refresh different components, each of them implementing `Events.UpdateFavoritesQuantity`.

For more information, read: <http://wiki.genexus.com/commwiki/servlet/wiki?31167>

GeneXus™

The power of doing.

Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications