

# Events in a panel object

## Client-side and server-side events

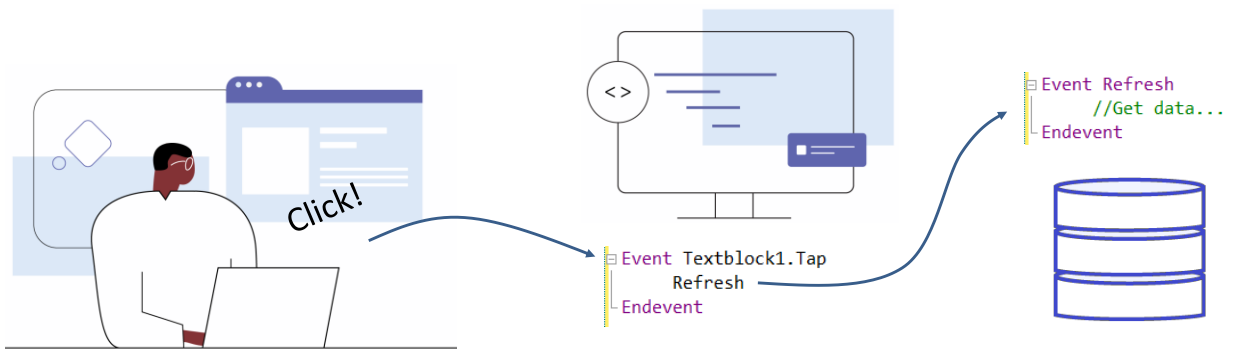


Panel objects are used to implement both web applications generated in Angular and applications generated for mobile devices in Android or iOS.

That's why the events available in this object are mostly valid for both platforms.

In this video, everything about events in a panel object will be applicable to both types of platforms. Exceptions will be explicitly made where appropriate.

## Use of events

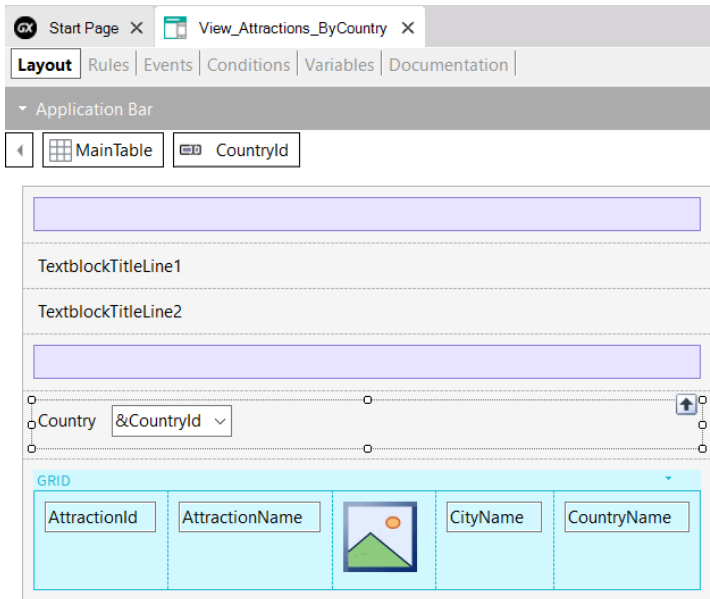


As we already know, events are messages displayed by the software or the system at certain moments during the execution of an application. This allows us to schedule actions to be executed at those moments.

For example, if the user clicks on a button or types something in a text control and exits it, or if the server is required to send updated information, we can schedule a response after the event is triggered.

Suppose that when viewing data about attractions, we would like to have filters so that the data displayed meets the selected constraint.

## A simple example



To implement this, we open the View\_Attractions panel and save it with the name View\_Attractions\_ByCountry.

Now we go to the variables section and add a variable &CountryId that automatically takes the type of the CountryId attribute.

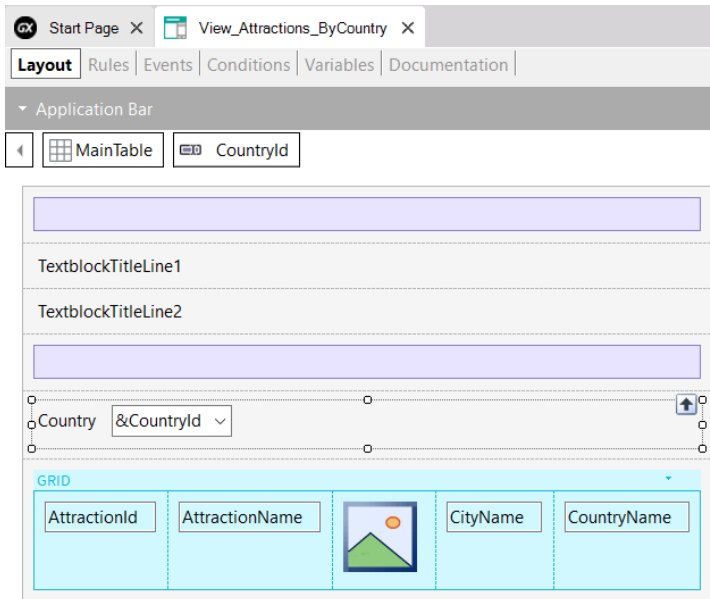
We drag it to the form after the textblocks of the title and in its properties we change the CountryId label to Country. Next, we change the ControlType property to Dynamic Combo and set Item Descriptions to CountryName. In addition, we set the EmptyItem property to True, so that at the beginning the combo appears without any default value.

For the filter to have an effect on the attractions that we see in the grid, we edit the grid properties and in the Conditions property we write: CountryId = &CountryId when not &CountryId.IsEmpty() and close with a semicolon. Now that we are in the grid properties, we set the Auto Grow property to True so that the grid auto-adjusts to show all the records.

Once we choose a country, the panel must be refreshed so that the grid loads only the attractions from that country.

For this we use the ControlValueChanged event of the dynamic combo, so we go to the Events tab and in the menu we choose Insert event, click on &CountryId, and select the event we mentioned.

## A simple example (continued)



As we can see, the event code opens for us to write what we want to have executed when this event is fired. We type Refresh. This command will cause the grid to get the data again, filtered this time by the chosen country.

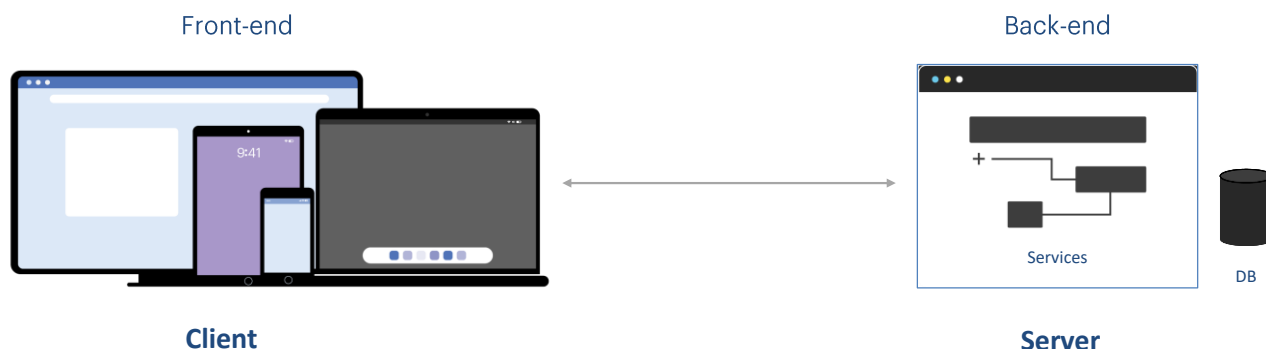
For the server to understand that we want to bring new data, the URL sent to the server must be changed, so that the server interprets that it is a new page and obtains the corresponding information to send it to the client. To this end, we add a Parm rule with the variables that we use in the filters, so that when they change their value the page is updated. In this case, we add only the variable &CountryId.

We right click on the View\_Attractions\_ByCountry panel and choose Run.

Since we had set the EmptyItem property to True, the combo is displayed without a selected country and all the attractions are shown. If we choose China, we see that the grid is reloaded and now shows only the tourist attractions of China.

Let's take a closer look at the events.

## Eventos del cliente y del servidor



- ClientStart
- Back
- User Events
- Controls Events
- WorkWith Predefined Events
- Only for mobile devices:
  - Navigation.Start Events

- Start
- Refresh
- Load

In a panel object, we have two types of events, events that are fired on the client side and events that are fired on the server.

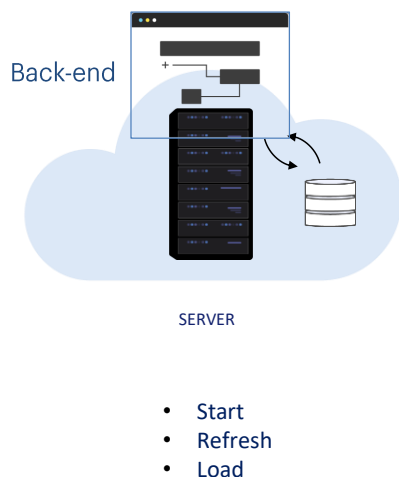
The server events are the same as when we work with webpanels, the Start, Refresh and Load events.

These events will always be fired on the server in the case of Angular applications and native online mobile applications. In the event that we are deploying offline native mobile apps, they will be fired internally on the device.

The events on the client side are ClientStart, Back, user-defined events, events associated with screen controls, and also the events predefined by the WorkWith pattern, which is a pattern that, when applied, allows generating a series of panels. that fulfill the functionality of working with an entity (to perform insertions, modifications, deletions, data navigation, filters, etc.), analogous to the WorkWith pattern that we use when working with webpanels.

In the event that we use the panel objects to generate an application for mobile devices, we will have the events associated with the navigation styles of the information, which depend, for example, on the type of device and its orientation when the application starts.

## Eventos del lado del servidor



- Start Event execute only once.
- Refresh Event is executed after Start Event
- Refresh Event trigger Load Event
- Load Event is the last of system Events executed (only if a grid exist)
  - Grid with Base Table: Load is executed as many times as records in base Table.
  - Grids without Base Table: Load is executed only once.
  - SDT based Grids: Load is not triggered.

Now, let's see the server-side events.

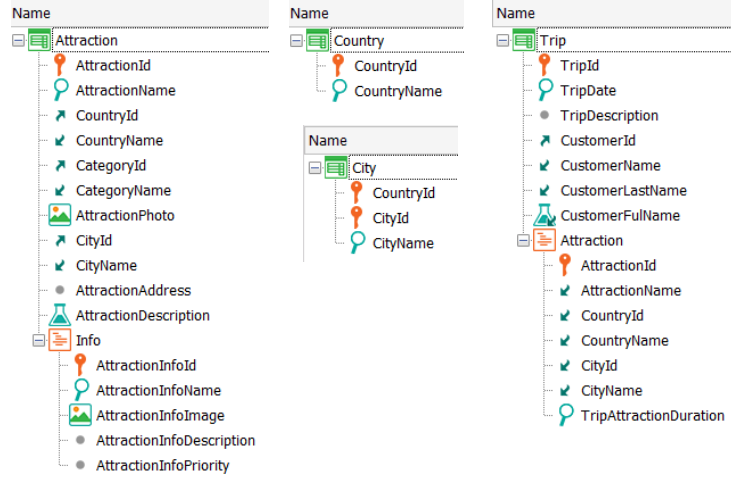
- The first one executed is the Start event. It is executed only once when the panel is opened, and will not be executed again unless we exit the object and reopen it.
- The Refresh event is executed after the Start Event, usually only once, but it can be invoked again using the Refresh command. In that case, it will be executed more than once and will be the first event, since Start will not be executed again.
- When the Refresh event is invoked, the Load event will be triggered at the end. This will only happen if there is at least one grid in the panel and the grid is not a collection SDT variable.
- The Load event is the last one of the system events to be executed:
  - If it has a base table, it will be executed as many times as records exist in the base table.
  - If the grid doesn't have a base table, it will be executed only once.
  - And if the grid is based on an SDT, the load event will not be executed.

In the case of an application for mobile devices, the code of the Start, Refresh and Load events has no access to the device's resources, such as camera, GPS, etc.

## Ejemplo de eventos del lado del servidor

The screenshot shows a web application interface with the following elements:

- Navigation tabs: Start Page, View\_Attractions\_MoreInfo\*
- Menu: Layout\*, Rules, Events\*, Conditions, Variables, Documentation
- Application Bar: MainTable, Grid1
- Form fields:
  - Country: &CountryId (dropdown)
  - Attraction Name From: &AttractionNameFrom
  - Attraction Name To: &AttractionNameTo
- GRID: A table with columns: AttractionId, AttractionName, AttractionPhoto, CityName, CountryName, &Trips.
- Summary: Total Trips: &TotalTrips
- Footer: Any Platform, Default Orientations, Add Layout, Delete Layout



## Editing Conditions

```
CountryId = &CountryId when not &CountryId.IsEmpty();
AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty();
AttractionName >= &AttractionNameTo when not &AttractionNameTo.IsEmpty();
```

Let's consider the case of the following panel, which allows viewing a grid of attractions and filtering the grid by country and attraction name.

The grid contains the attributes AttractionId, AttractionName, AttractionPhoto, CityName, CountryName and a &Trips variable that will show the trips made to each attraction.

Below the grid, we can see the overall total of trips made to all attractions.

We can also see the model of the transactions used in this panel and the grid conditions that allow performing the filters.

## Ejemplo de eventos del lado del servidor

The screenshot displays the GeneXus IDE interface. On the left, a UI layout is shown with a grid containing columns for AttractionId, AttractionName, CityName, CountryName, and Trips. Below the grid is a 'Total Trips' label. The right pane shows the 'Events' tab with the following code:

```

1 Event ClientStart
2   TextblockTitleLine1.Caption = "The new age of"
3   TextblockTitleLine2.Caption = "EXPLORATION"
4 -Endevent
5
6 Event Load
7   &Trips = Count(TripDate)
8 -Endevent
9
10 Event Refresh
11   &TotalTrips = 0
12   For each Trip.Attraction
13     Where CountryId = &CountryId when not &CountryId.IsEmpty()
14     Where AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty()
15     Where AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty()
16       &TotalTrips += 1
17   Endfor
18 -Endevent
19
20 Event &CountryId.ControlValueChanged
21   Refresh
22 -Endevent
23
24 Event &AttractionNameFrom.ControlValueChanged
25   Refresh
26 -Endevent
27
28 Event &AttractionNameTo.ControlValueChanged
29   Refresh
30 -Endevent

```

The bottom pane shows the 'Editir' window with the following filter expression:

```

CountryId = &CountryId when not &CountryId.IsEmpty();
AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty();
AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty();

```

The Refresh and Load events that are triggered on the server side are programmed in the events tab.

In the Load event, we calculate the total trips of an attraction by means of a formula that accesses the TripAttraction table, corresponding to the second level of the Trip transaction. Note that in the Count formula we are referring to the TripDate attribute, and due to the attributes present in the grid, the base table of the grid (that is, of the variable part of the panel) will be TripAttraction, so the trips corresponding to each attraction will be counted.

In the Refresh event, we program a For Each to access the TripAttraction table to count the overall total of trips of the attractions. As a result, when the fixed part is drawn, the data corresponding to the total number of trips is already available.

*[If it were a Web Panel, it would be done in the Load event, but since it is a Panel, it would be done in the Refresh event]*

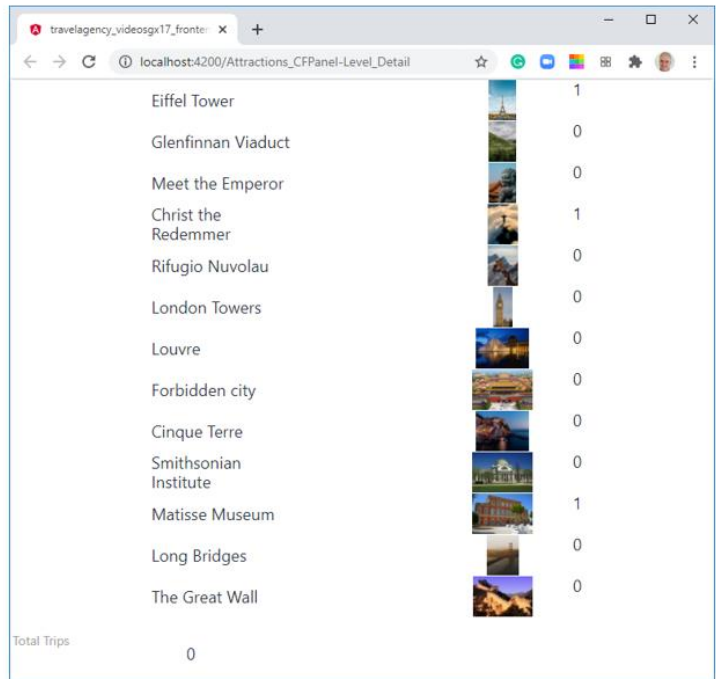


## The overall total of trips is miscalculated!

```

Layout * | Rules | Events * | Conditions | Variables |
Events
1 Event Load
2   &Trips = Count(TripDate)
3   &TotalTrips = &TotalTrips + &Trips
4 -Endevent
5
6 Event Refresh
7   &TotalTrips = 0
8 -Endevent
9

```



[Let's see a snippet from the Panel Object video. first steps]

If we look at the application running in the browser, we see that the total number of trips of each attraction is displayed correctly, but the overall total of trips comes out at 0.

What did we do wrong? The `&TotalTrips` variable is being increased in the Load event as we did in the web panel and we are sure that this event is being triggered because we see that some attractions have trips... So?

The reason is that panel objects do not work in the same way as web panels. In panel objects, the fixed part is loaded independently of the grid.

In this case, the `&TotalTrips` variable is in the fixed part of the panel, which is the first thing to be loaded. Then the grid is loaded, so when the variable is displayed, the grid has not been loaded, the Load event has not been triggered, and the value of `&TotalTrips` has not been added yet.

Remember that in a panel object used to develop customer-facing applications, to load the screen in the client device, services located in the server are invoked which are the ones that access the database. These services are data providers, which are independent for the fixed part and for the grid (or each grid) of the screen.

When the panel starts running, a local event is triggered on the client that invokes the data provider to load the fixed part and causes the Start and Refresh events to be triggered on the server. Then a second data provider is executed which triggers the Load event on the server N times and the grid is loaded.

In our example, as the Refresh is triggered first, the &TotalTrips variable is initialized and the fixed part is loaded; later, the Load event is triggered which is where &TotalTrips is loaded with the correct value and the grid is updated, but the fixed part was already loaded before and is not redrawn.

Due to this feature we cannot program the panel object as if it were a web panel.

In another video we will have a closer look at the triggering of events and the determination of base tables; for now, to make the example work we will change the programming.

## Correct Solution

```
1 | Event Load
2 |     &Trips = Count(TripDate)
3 | Endevent
4 |
5 | Event Refresh
6 |     &TotalTrips = 0
7 |     For each Trip.Attraction
8 |         &TotalTrips += 1
9 |     Endfor
10| Endevent
```

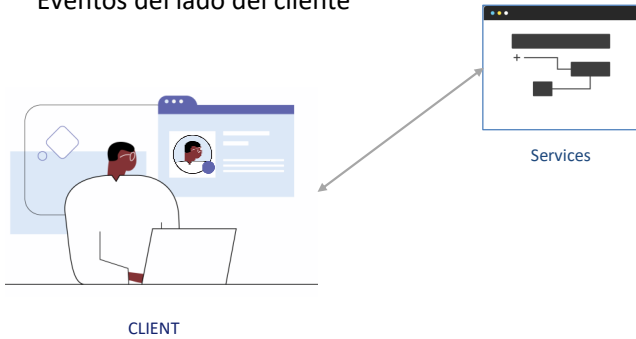
The solution is to include in the Refresh event a For Each command that accesses the TripAttraction table and counts the overall total of trips. Let's not forget to remove the calculation we had in the Load event.

The reason why we include the update in the Refresh event, using a For Each command, is because the Refresh event will be triggered when the fixed part is loaded, which will be before the grid is loaded.

Therefore when loading the fixed part, the value of &TotalTrips will have the correct value and only after that the grid part will be updated.

*[End of review]*

## Eventos del lado del cliente



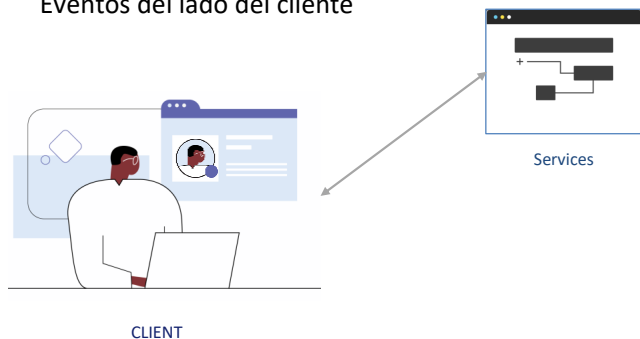
## • Types:

- System Events: ClientStart, Back
  - User Events
  - Screen Controls Events
  - Others: WW events & Navigation.Start
- Call to services to access server resources
  - Do not trigger Server Side Events (unless Refresh command is used)
  - In mobile apps, Access to devices resources
  - Use different Grammar & use of Composite

Now, let's see Client-side events. These events are the application's response to user interaction.

- There are several types of client events: system events such as ClientStart and Back, user events, on-screen control events and others that we will see next.
- The code associated with these events is executed in the client, unless access to a server resource is requested; for example, if you want to access the database. In this case, the client will have to invoke a server service.
- During the execution of a client-side event, server-side events are not executed unless explicitly requested through the Refresh command. This causes the server's Refresh event to be triggered, followed by the Load event (if there is at least one grid, as we saw before).
- When running a native application for mobile devices, client-side events have access to all hardware and software resources of the device, such as camera, GPS, microphone, calendar, contacts, etc.
- These client-side events will have a specific grammar that is different from that of server-side events. In particular, when it comes to concatenating more than one action in an event, a special command called Composite must be used that will be discussed later.

## Eventos del lado del cliente



- ClientStart
- Back
- User Events
- UI Controls Events
- WorkWith Predefined Events
- Only for mobile devices:
  - Navigation.Start Events

Let's take a brief look at each client-side event.

The **ClientStart** event is the first event to be triggered, even before the Start event that is triggered on the server and with no need for any user interaction with the application. It is used to initialize the home screen and UI-related aspects.

The **Back** event is used on mobile platforms and allows you to capture that the user has pressed the back button on Android devices, or that the back gesture was made on iOS devices, and program an appropriate action.

**User** events have a name given by the user and allow the developer to associate a certain code to be executed when a certain on-screen control is activated by clicking (or tapping) on it.

**On-screen controls** have their own events, depending on the control involved, such as: click (or tap), double-click (or double-tap), drag, swipe, ControlValueChanged, etc. These events are triggered when some of the above actions take place and the developer can program a response of the application to the user's interaction.

The **WorkWith** pattern has predefined events that are triggered depending on the action performed on the data of the entity to which the pattern is applied. These include the Insert, Update, Delete, Save, Cancel events, among others. The events available will depend on the part of the WorkWith object being executed (list, detail, etc.).

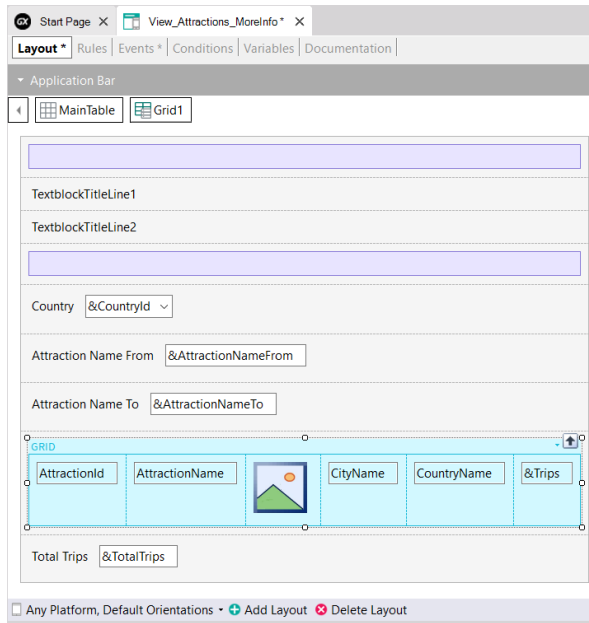
In the case of mobile devices, depending on the type of device and orientation, when we start the application a Start event will be triggered depending on the navigation. For example, if we are using a Tablet, by default the application starts in Split mode, meaning that the screen is divided into two sections, with a list on the left and the detail of the selected item on the right. In this case, when the application starts, the Split.Start event is triggered.

In the case of phones, for example, the screen will show only the list and a separate screen will be opened to view the details. This mode is called Flip and is the default behavior in these devices. Therefore, when starting the application, the Flip.Start event will be triggered.

Although there is a default navigation for each device, the main objects of mobile applications have a property that allows changing the way the application starts.

These Start events associated with the type of navigation are triggered at the start of the mobile application and immediately after the ClientStart event.

## Ejemplos de eventos del lado del cliente



```

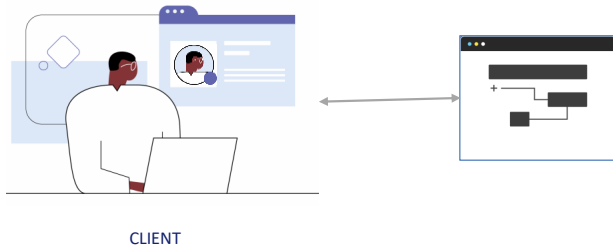
Rules | Events | Conditions | Variables | Documentation
1 | Event ClientStart
2 |     TextblockTitleLine1.Caption = "The new age of"
3 |     TextblockTitleLine2.Caption = "EXPLORATION"
4 | Endevent
5 |
6 | Event Load
7 |     &Trips = Count(TripDate)
8 | Endevent
9 |
10 | Event Refresh
11 |     &TotalTrips = 0
12 |     For each Trip.Attraction
13 |         Where CountryId = &CountryId when not &CountryId.IsEmpty()
14 |         Where AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty()
15 |         Where AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty()
16 |             &TotalTrips += 1
17 |     Endfor
18 | Endevent
19 |
20 | Event &CountryId.ControlValueChanged
21 |     Refresh
22 | Endevent
23 |
24 | Event &AttractionNameFrom.ControlValueChanged
25 |     Refresh
26 | Endevent
27 |
28 | Event &AttractionNameTo.ControlValueChanged
29 |     Refresh
30 | Endevent

```

In the object we saw before, we program only events associated with controls, we don't program any user events.

To each variable of the on-screen filters, we program its ControlValueChanged event to trigger the Grid Refresh method, which will cause the server Refresh and Load events to be triggered to update the contents of the grid with the filters entered.

## Qué podemos hacer en eventos en el cliente



- Call to other panels
- Call to Rest Services
- Use of Business Component
- Call to WorkWith objects. In mobile call to Menu
- Call Externals Objects of GeneXus module For Angular apps see: <https://wiki.genexus.com/commwiki/servlet/wiki?46456>
- Call Subroutines
- We can do:
  - Control properties assignments
  - Simple or SDT variable assignment
  - For each Line and Selected Line in grids
  - Use If-Else, Do-Case and Do-While code blocks.

In short, let's see what can be done in a client-side event:

- Invoke another panel object.
- Invoke Rest Services and when we invoke Procedures or Data Providers, they will automatically be exposed as Rest Services on the server. If those procedures or DPs were invoked only from a server-side event (within Start, Refresh or Load events), they would not be exposed as REST services because they would not be necessary.
- Use Business Components to retrieve or update information. In this case, these Business Components will also be exposed as Rest services automatically.
- Directly invoke any node of the Work With pattern. In mobile applications, we can invoke a Menu object.
- Invoke the external objects defined in the GeneXus module. Some of these APIs only make sense for a particular platform, such as mobile devices. Meanwhile, some of them can only be used on web applications, and others can be used on both platforms. To find out which ones can't be accessed by Angular applications, visit the wiki page shown on screen (<https://wiki.genexus.com/commwiki/servlet/wiki?46456>)
- Call subroutines.
- In addition, we can:
  - Assign properties to controls
  - Assign simple or SDT variables
  - Run For Each Line and For each Selected Line on grids
  - Use the blocks IF-Else, Do-Case and Do-While

If we try to use commands not allowed in a client-side event, we will see an error in the output screen for those lines that don't comply with the grammar constraints. For example, if we try to use a For Each command, a New or any attempt to access or modify information that is only accessed from the server.

## Gramática de eventos del lado del cliente

Composite

<Control>.<Property> = <value>

If <Bool\_expr>

Do case... endcase

Do while <Bool\_expr>

Do-sub (except Menu for Smart Devices)

For each selected line

Simple variable assignment: &var = <expr>

SDT or BC elements assignment:

&SDT.A = <value>

&BC.A = <value>

Return

Refresh

## COMMANDS

Inside an **expression**:

Variables

Attributes

Constants

Methods

Functions

Control properties

Operators (+, -, /, ^)

Here is a summary of the commands we can use in the client-side event code.

These restrictions are only for client-side events, but not for server-side events (Start, Refresh, and Load) where we can use all commands and functions available in GeneXus.

As for commands, the accepted ones are shown here.




## Comando Composite

Country

Name From

Name To

GRID

AttractionId	AttractionName	CountryName		&Trips	&Detail

Total Trips

Attractions report

```

1 Event "Attractions report"
2   Composite
3     AttractionsByName("C", "M")
4     Return
5   EndComposite
6 Endevent

```

- Only for Client Side Events with more than 1 line of Code.
- Stop execution on Error.
- Automatic Error Handling.

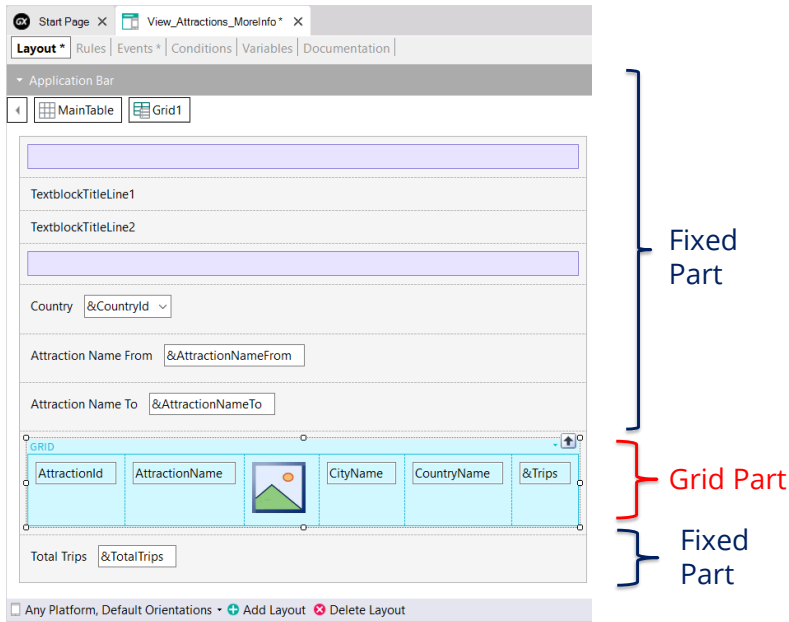
Now, let's look at the composite command mentioned above.

This command is used in panel objects on client-side events, when two or more lines must be written in an event; in this case, it is mandatory to group the complete event code inside this command.

This command is important because if an error occurs in the call sequence, it stops running and the errors are automatically handled and displayed on the screen with no need to implement any programming.

It is a great difference with web panels, because when a called object in an event causes an error, the execution is not interrupted—it continues in the following statement and the developer is responsible for handling the errors and programming the actions to be taken.

## Orden de ejecución de los eventos



- ClientStart
- Only for mobile: Navigation.Start
- *Call to Fixed part Data Provider*
  - Start event
  - Refresh event
- **Fixed Part is Drawn**
- *Call to Grid Data Provider*
  - Load event
- **Grid Part is Drawn**

Now let's see what happens with events when we run a panel object.

The ClientStart event is executed first, and it is executed only once on the client. In mobile applications, the Start event corresponding to the type of navigation set in the main object is then triggered by means of the Navigation Style property.

Next, a data provider is executed that will return the data needed to load the fixed part of the panel. This data provider is involved in the execution of the code of the Start and Refresh events to be executed on the server and returns, in a single result, the information to load the fixed part.

The fixed part of the panel is drawn later.

Then, a second Data Provider is executed that will retrieve the data required by the grid. Within the execution of this data provider, the Load event code is executed on the server. This Load event will be executed N times when the grid has a base table, once for each record, only once if the grid doesn't have a base table, and not at all if the grid was from a collection SDT variable.

At the end of the execution of the data provider, it will return the information generated by all these executions of the Load event in a single result, which will be used to load the grid and then finish drawing the grid.

## Uso del comando Refresh en eventos del cliente

The image shows a GeneXus application interface and its rules. The interface includes:

- TextblockTitleLine1 and TextblockTitleLine2.
- Country dropdown (&CountryId).
- Attraction Name From (&AttractionNameFrom) and Attraction Name To (&AttractionNameTo) filters.
- A GRID with columns: AttractionId, AttractionName, CityName, CountryName, and &Trips.
- Total Trips (&TotalTrips) counter.

A blue arrow labeled "Parte fija" points to the Country, Attraction Name From, and Attraction Name To filters. Another blue arrow points to the &Trips column in the grid.

The rules section shows the following code:

```

1 | Parm(&CountryId, &AttractionNameFrom, &AttractionNameTo);
2 |
3 |
4 |
5 |
6 | Event ClientStart
7 |   TextblockTitleLine1.Caption = "The new age of"
8 |   TextblockTitleLine2.Caption = "EXPLORATION"
9 | Endevent
10 |
11 | Event Load
12 |   &Trips = Count(TripDate)
13 | Endevent
14 |
15 | Event Refresh
16 |   &TotalTrips = 0
17 |   For each Trip.Attraction
18 |     Where CountryId = &CountryId when not &CountryId.IsEmpty()
19 |     Where AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty()
20 |     Where AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty()
21 |     &TotalTrips += 1
22 |   Endfor
23 | Endevent
24 |
25 | Event &CountryId.ControlValueChanged
26 |   Refresh
27 | Endevent
28 |
29 | Event &AttractionNameFrom.ControlValueChanged
30 |   Refresh
31 | Endevent
32 |
33 | Event &AttractionNameTo.ControlValueChanged
34 |   Refresh
35 | Endevent

```

We've mentioned that the fixed part of the panel is loaded independently from the loading of the grid. Different data providers are invoked that are published on the server as services and access the database to retrieve the information of each part.

When we change the value of a filter, the grid's information needs to be refreshed. For this, we need to add the grid's Refresh method, which will trigger the server's Refresh and Load events. In turn, it will cause the grid to be reloaded, applying the programmed conditions and displaying the filtered results as expected.

Since the grid's Refresh command must be invoked after we change the value of the filter variable, we use the ControlValueChanged event of each variable to invoke the method. In this way, after changing a filter value, when leaving the field the corresponding event will be triggered, which is expected to refresh the content of the grid.

But what happens when the Refresh command is invoked within a client-side event?

In this architecture, since the objective is to have the page loaded as few times as possible, the data cache is prioritized; that is to say, we always try to retrieve the information previously saved. That is, depending on the caching configuration, it is decided whether or not to go to retrieve data from the server.

When it is decided to go to the server, if there are no changes in the server data, nothing is returned to the client.

Otherwise, the Refresh and Load events are executed (if there is a grid not based on an SDT, as in our case) and the fixed and variable parts of the panel are updated, as expected.

For the server to understand that we want to bring new data, the URL sent to the server must be changed, so that it understands that it is a new page and obtains the corresponding information to send to the client.

To do so, we add a Parm rule containing the values of the variables used in the filters. In this way, if a variable's value changes, the page is updated with the new data.

We run the panel to test what we have seen.

## Uso del comando Refresh en eventos del cliente (cont.)

The screenshot displays the GeneXus IDE interface. On the left, a visual representation of a user interface is shown, including text blocks, a dropdown menu for 'Country', two text input fields for 'Attraction Name From' and 'Attraction Name To', a grid, and a 'Total Trips' field. A blue arrow labeled 'Parte fija' points to the '&TotalTrips' field in the grid. On the right, the 'Rules' pane shows the code for the 'Refresh' event of the '&AttractionNameFrom' control. The code includes logic for calculating the total number of trips for each attraction and updating the grid.

```

1 | Parm(&CountryId, &AttractionNameFrom, &AttractionNameTo);
2 |
3 |
4 |
5 |
6 | Event ClientStart
7 |   TextblockTitleLine1.Caption = "The new age of"
8 |   TextblockTitleLine2.Caption = "EXPLORATION"
9 | Endevent
10 |
11 | Event Load
12 |   &Trips = Count(TripDate)
13 | Endevent
14 |
15 | Event Refresh
16 |   &TotalTrips = 0
17 |   For each Trip.Attraction
18 |     Where CountryId = &CountryId when not &CountryId.IsEmpty()
19 |     Where AttractionName >= &AttractionNameFrom when not &AttractionNameFrom.IsEmpty()
20 |     Where AttractionName <= &AttractionNameTo when not &AttractionNameTo.IsEmpty()
21 |     &TotalTrips += 1
22 |   Endfor
23 | Endevent
24 |
25 | Event &CountryId.ControlValueChanged
26 |   Refresh
27 | Endevent
28 |
29 | Event &AttractionNameFrom.ControlValueChanged
30 |   Refresh
31 | Endevent
32 |
33 | Event &AttractionNameTo.ControlValueChanged
34 |   Refresh
35 | Endevent

```

With empty filters, all the attractions are shown. For each attraction, the total number of trips—calculated in the Load event that updated the grid information—is correctly displayed.

In addition, the overall total of trips that is in the fixed part is shown correctly, since it was calculated in the Refresh event and was executed before the fixed part was drawn.

If we now choose the attractions of the country France and choose to see the attractions starting with the letter A to the letter N, the value of the filters included in the grid conditions works correctly. Also, the invoked Refresh commands caused the Refresh and Load events to be triggered, so the information was correctly updated when retrieved from the server and we only see the attractions of France with the name in the chosen range.

The following videos will discuss other aspects of the design and implementation of panel objects.

# GeneXus™

[training.genexus.com](http://training.genexus.com)

[wiki.genexus.com](http://wiki.genexus.com)

[training.genexus.com/certifications](http://training.genexus.com/certifications)