Global Events

As we know, the use of components allows us to reuse functionalities that have already been developed.

For example, in the case of a web panel like the one we have seen where the tourist attractions that a travel agency offered to visit were listed. We will not deal here with the design, which can certainly be improved.

We are looking at a rather simple screen, which shows a fixed part that comes from the master page, and another part of its own. We can filter by country, and by name of attraction.

However, what matters right now is to note that we have added the possibility of marking tourist attractions as favorites. For example, I mark this one and this one, so we have marked two attractions as favorites.
However, if we look up here, there is a zero, as if we hadn't marked any of them as favorites.

Let's refresh the screen... and now we see a 2.
We also want that, if we unmark a favorite attraction, this number is updated to 1.
How do we get this value to update automatically depending on whether a tourist attraction is marked or unmarked as a favorite?

OK. Let's introduce the concept of global event.

In general, in interactive objects such as web panel objects, events are actions handled by the program and triggered by the user. Events are usually local to the program in which they are defined.

Global events, on the other hand, allow you to define events that apply to all components of an application.

While local events are triggered in response to a user action, global events are code that remains inactive until it is invoked from another event, from any other component.

In the image displayed, a global event defined in “Component E” could be invoked from “Component A.” Any combination is possible, regardless of the nesting level of the components that make up the screen.

Let's go to GeneXus to see this Web panel.

First, it has an associated master page, which is the default. Therefore, the content of its screen will be loaded inside the content placeholder of that master page.

In addition, we have added another component to show the total number of tourist attractions that the user has marked as favorites.

To save the attractions that a user has marked as favorites, we should have a Users table to indicate each user’s favorite attractions. If we don't want to work with users yet (for example, because we haven't applied GAM yet and we are still figuring out if we are going to have our own Users table or not) we can temporarily keep favorites per browser instance.

To this end, we have created this transaction with this attribute, to which we have specified this new domain that we have also created, with the same data type as this property...

ClientInfomation is an external object of the GeneXus module, whose ID property allows identifying the client device that is running, both natively and on the web.

In the case of web applications, the property returns a user ID, which persists between all sessions with the same browser and for the same application. So what we have done is to add a transaction with a key made up of the device and the tourist attraction. We will save the favorites in the table associated with this transaction.

In the Web component being loaded here, we see that it only contains a text block with this name, whose caption is loaded at runtime–only in the Start event, invoking this subroutine that first calculates the number of tourist attractions for this client, which are in the table containing these two attributes (FavoriteAttraction).

Here is the value we are looking for. Then we turn that numeric value into a string, to assign it to the textblock that will be displayed.

On the other hand, if we look at what will be loaded in the ContentPlaceHolder for our Web panel... we see that it is a screen with this combo box to choose a country, and below it we have another component, to which the value of that country is sent every time we modify it.

And if we look at what is being loaded in that web component, it is this other web object of the component type, which has a grid that filters the tourist attractions it will show, according to the country received by parameter and the screen's filters.

We have chosen a Flex grid, which contains a Canvas table, in order to overlap the tourist attraction photo with its name and a Boolean variable to allow the user to mark or unmark the tourist attraction as a favorite.

We will not dwell on the design of all this.

Just note that in the Load event of the grid, which has Attraction as its base table, this For each command is executed for every tourist attraction to be loaded. It searches in the subordinate table, FavoriteAttraction, if there is a record for this device, the one that is running.

Programming the Click event on this variable

Once the attractions grid is loaded, when the user clicks on an attraction favorite, we call a procedure that checks whether the attraction was already marked as a favorite; if so, it removes it from the table. And if it's the other way around, it does the opposite; that is, it inserts it.

When doing this, the total of favorite attractions should be updated.

In other words: we want a user event of a component loaded in this panel to perform an action on a component with which it has no other relationship than being indirectly involved in the same screen.

With this predefined object, Global Events, so that we can modify it.

We could save it under another name and create as many particular objects as we want.

In this case, we are going to modify the predefined one, which comes empty, adding an event that we will call in this way.

The events added will be able to handle parameters, if required for communication between components.

Here we only need one to find out that another one triggered it. No parameters are needed.

Since we have this global form of communication, we are going to use it.

Let's go to the panel that should trigger the event. It will happen every time the variable is clicked on. At that moment, we invoke the GlobalEvents external object: the only one there is at the moment. In this way, the triggering of the event is being reported.

to show attractions, it has to subscribe to this event, in order to listen to it whenever it takes place.

Commented [RP1]: En español dice: “atracciones para mostrarlas se suscriba a este evento, para poder escucharlo toda vez que se produzca.”

For this, we simply set to listen to the event. And there we program what we want to be executed when the event occurs, which in this case is to recalculate the number of attractions.

Refreshing. Now we mark, and mark... Communication is taking place successfully.

Even though here we have seen an example of communication between web panels with components, the same is true for communicating multi-experience panels with components, for example, for native applications.

To learn more about this topic, you can visit our wiki.