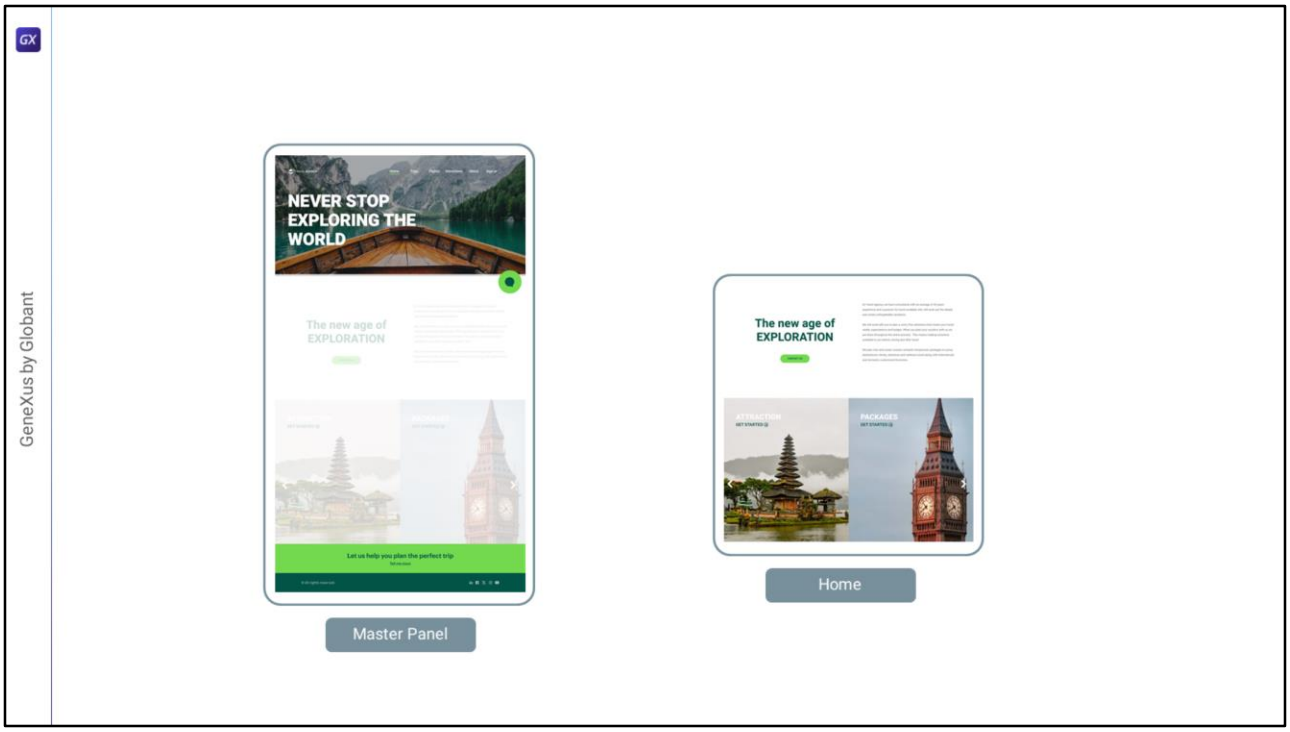
GeneXus by Globant

Master Panel:
Header update and Navigations

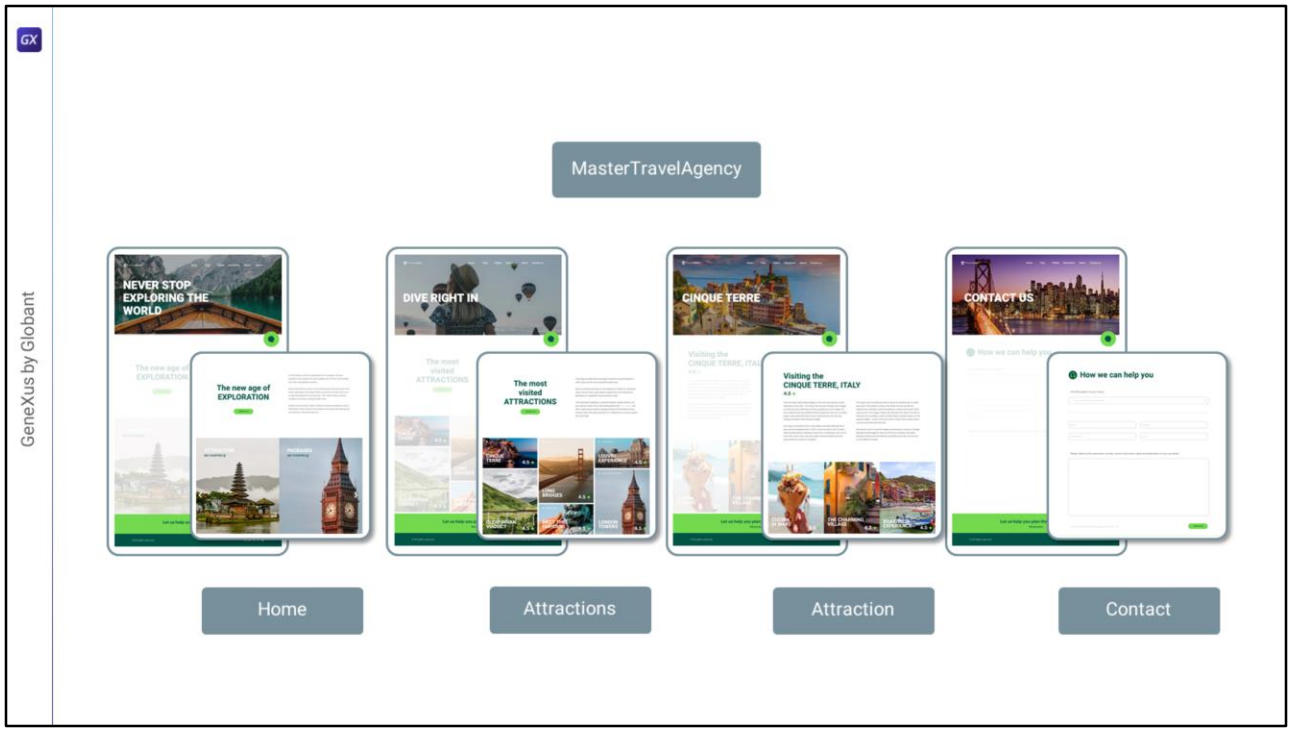
Cecilia Fernández



OK, in this video we are going to complete the implementation of the Header.

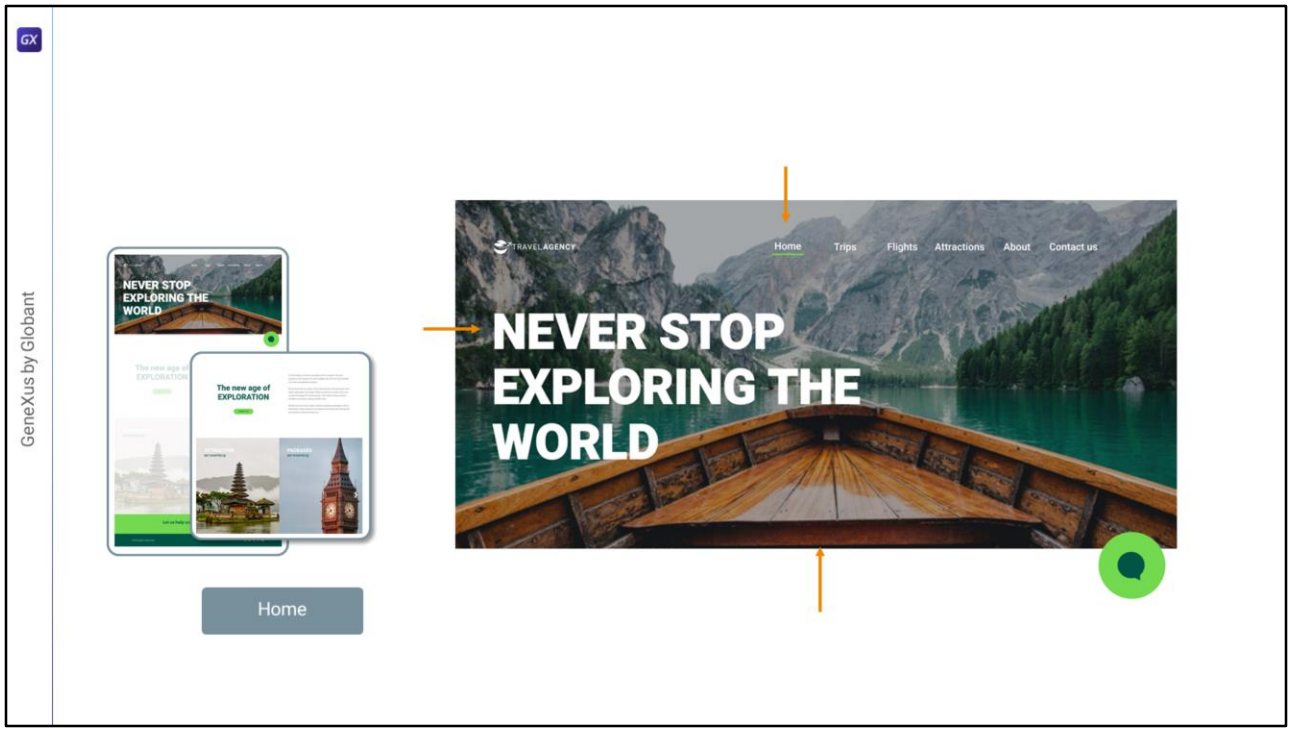
As we know, the Master Panel is not an object that can be executed independently. It is executed whenever a panel that has it as Master Panel is executed.

Also, the panel layout will be rendered inside the ContentPlaceHolder control of the Master Panel.

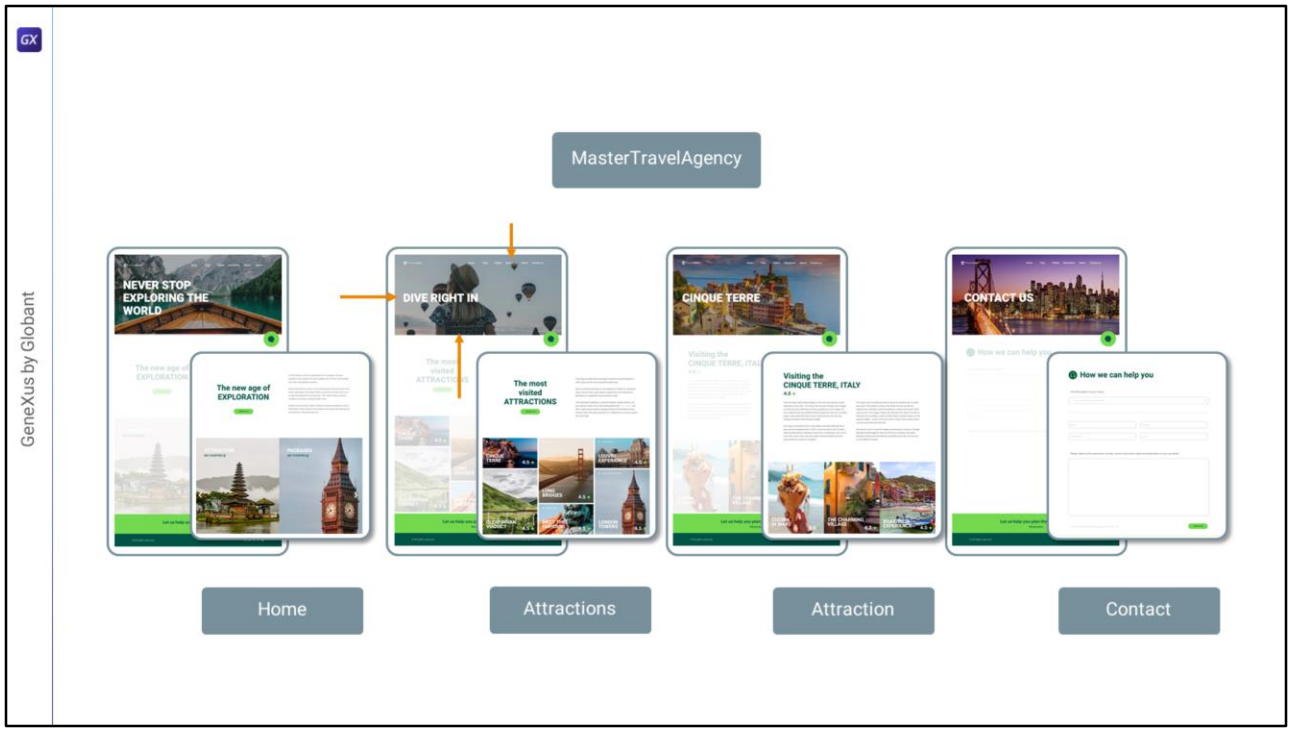


In our case, these four panels will be the executables. Every time one of them is invoked, it will execute the Master Panel.

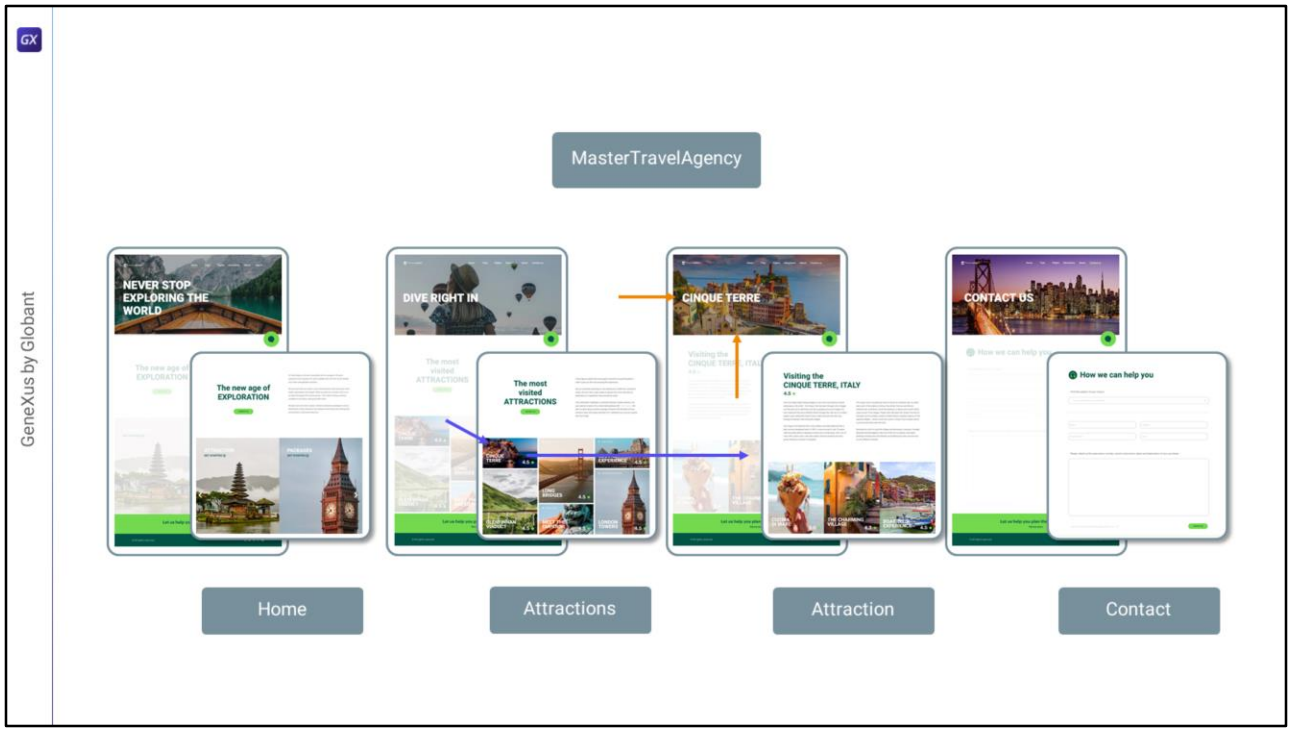
The Master Panel will have to know who made it run because this “who” determines three things: the background image of the Header, the title over the image, and the menu option that should be the selected one.



For Home, we should see this image and title, and the button that should be selected (with the green indicator below it) is Home.

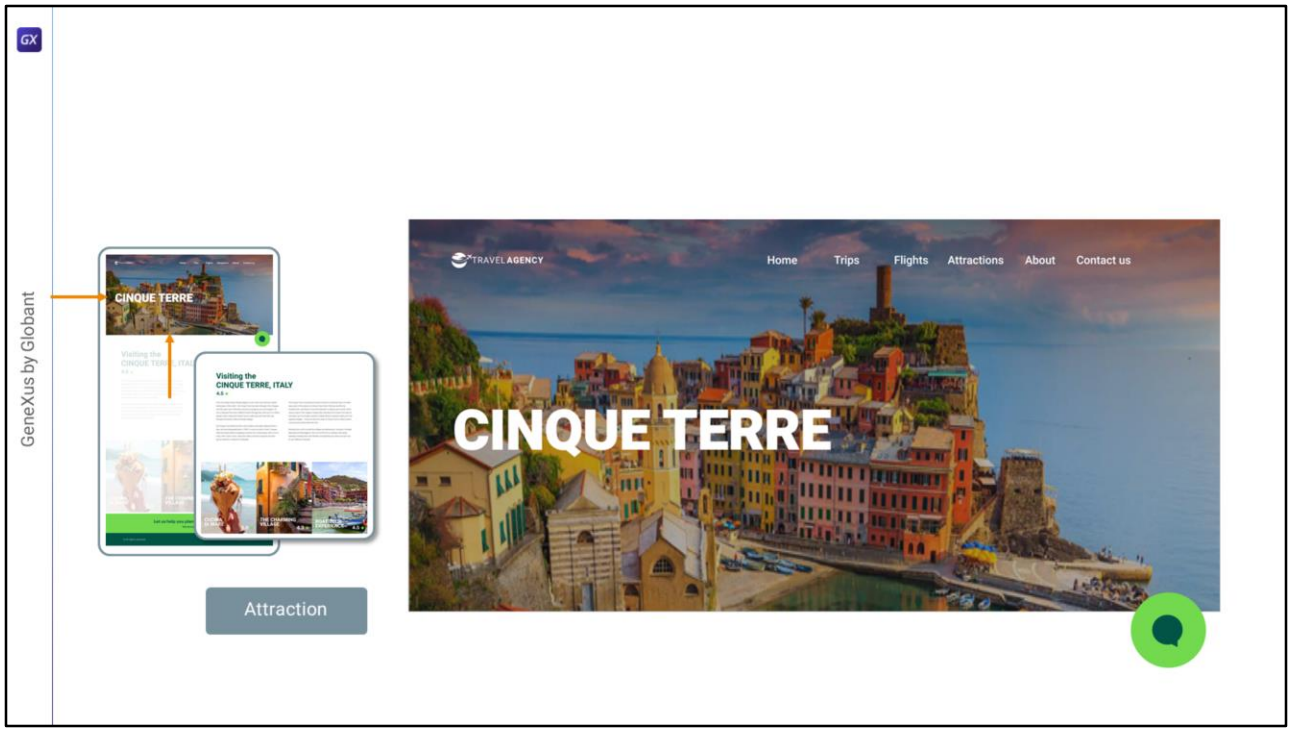


For Attractions, these should be the image and the title, and the selected button should be Attractions.

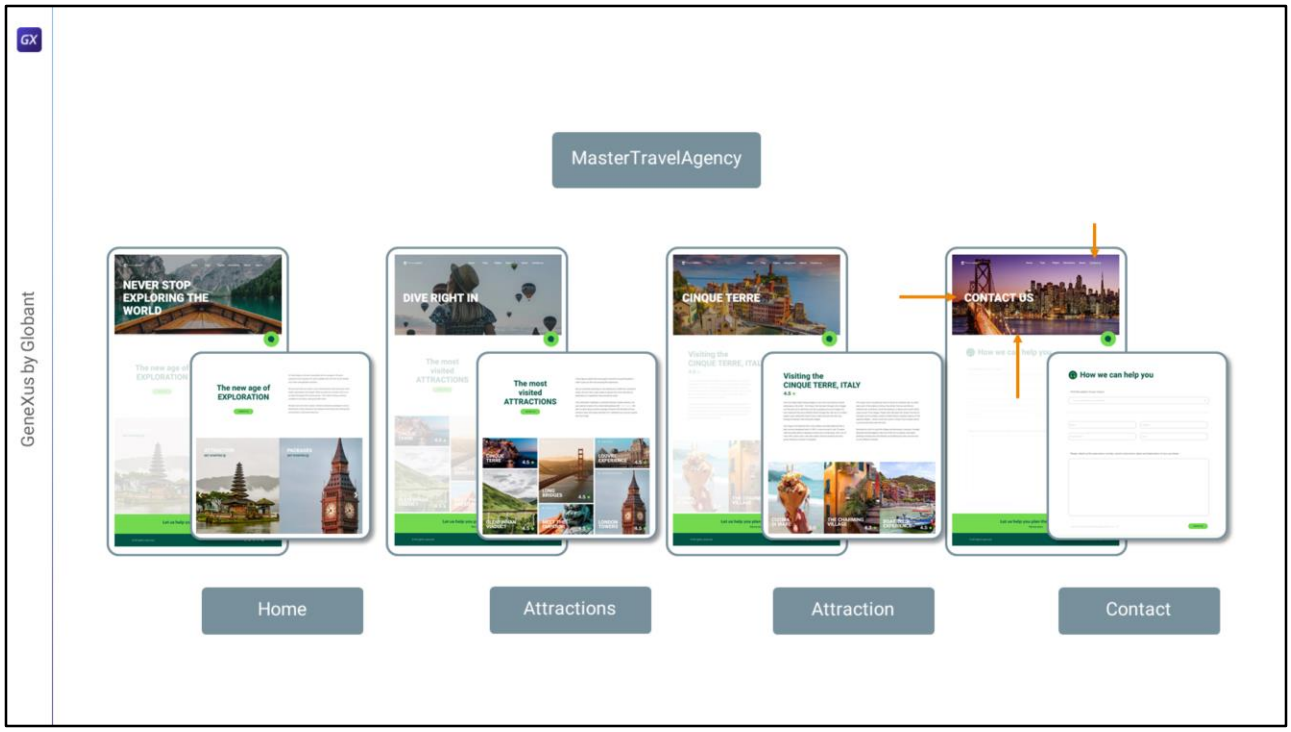


For Attraction something different happens because it is not a menu option. This page will be accessed from this other page, choosing one of the tourist attractions shown in this carousel. Then this panel will receive the tourist attraction identifier by parameter, and must access the database to load here the information of that tourist attraction.

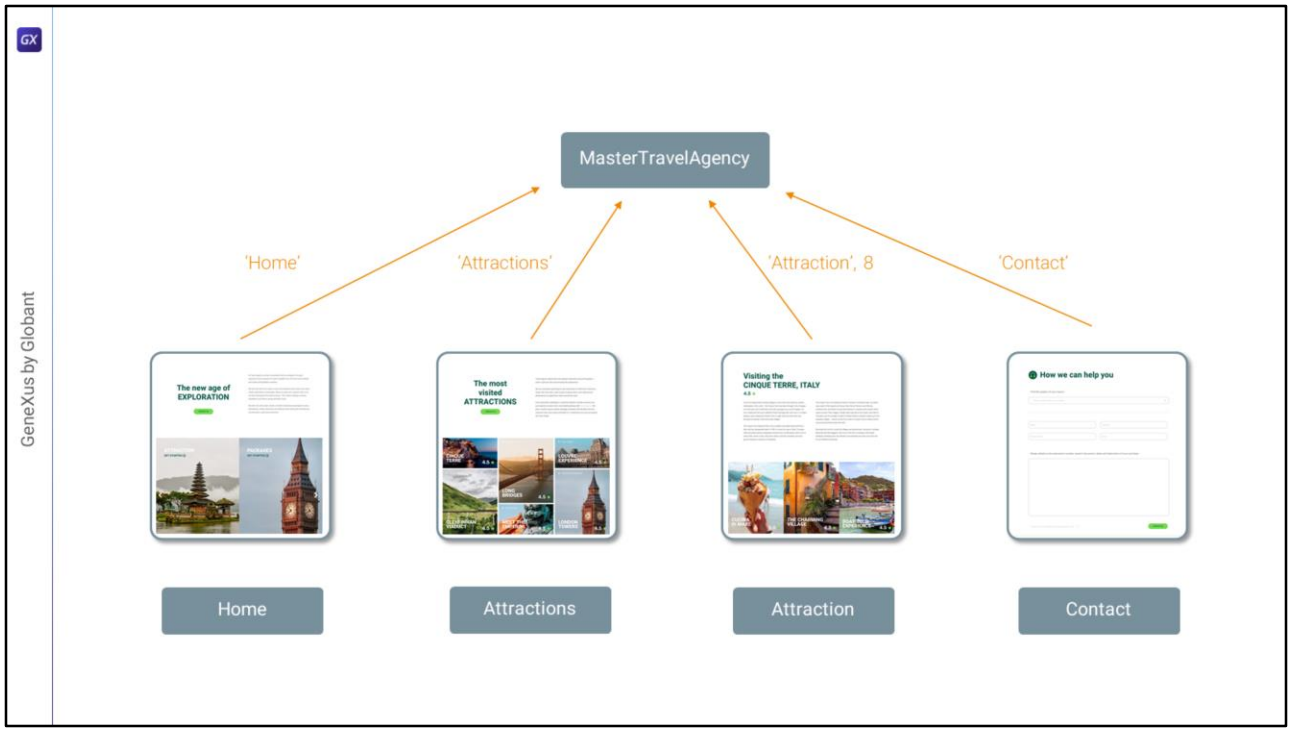
In addition, at the Master Panel level, it will also have to know what that tourist attraction is, because it will also have to go to the database to bring its photo, that it will show as Hero in the Header, and its name to show as title...



...and in the case of the menu, actually all options should be cleared.



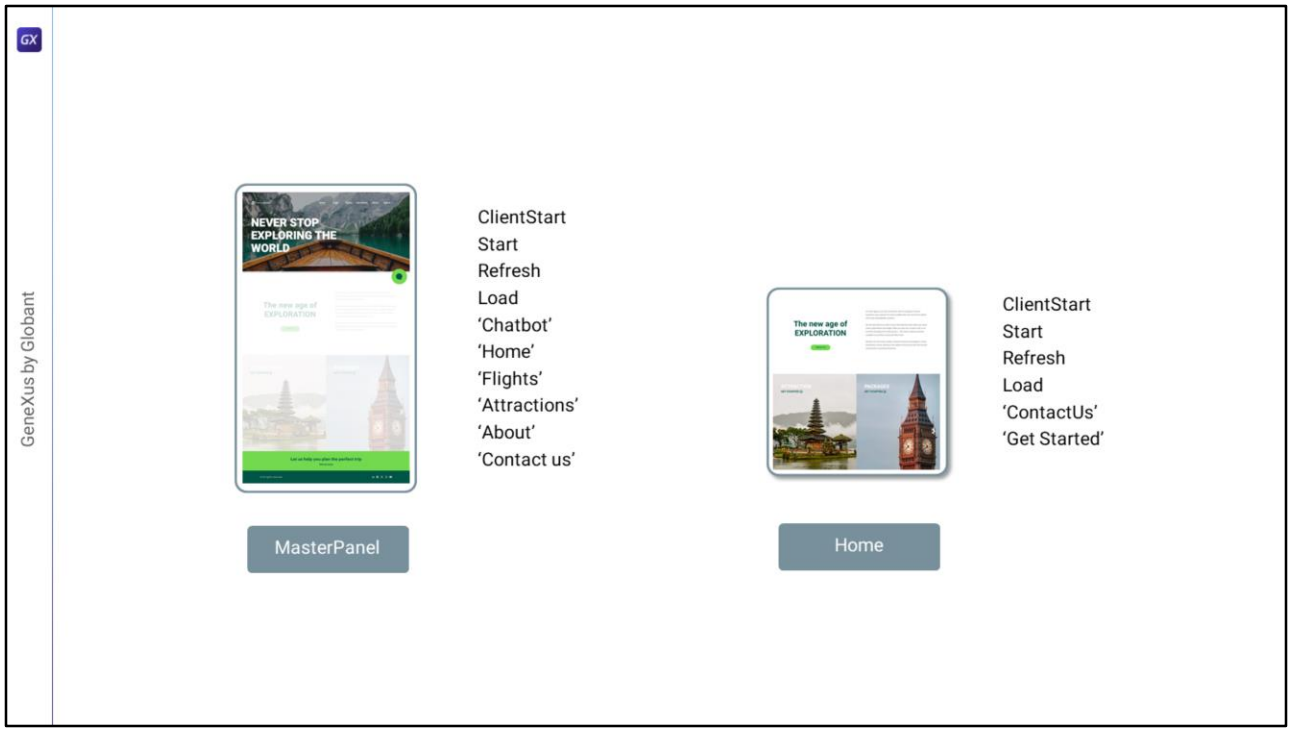
The contact panel is the same as the first two, with a fixed image, text and selected button.



In summary, the panel executed each time should be identified to the Master Panel, so that it can take the necessary actions on the 3 elements of the Header that we have just analyzed: background image, title, and selected button.

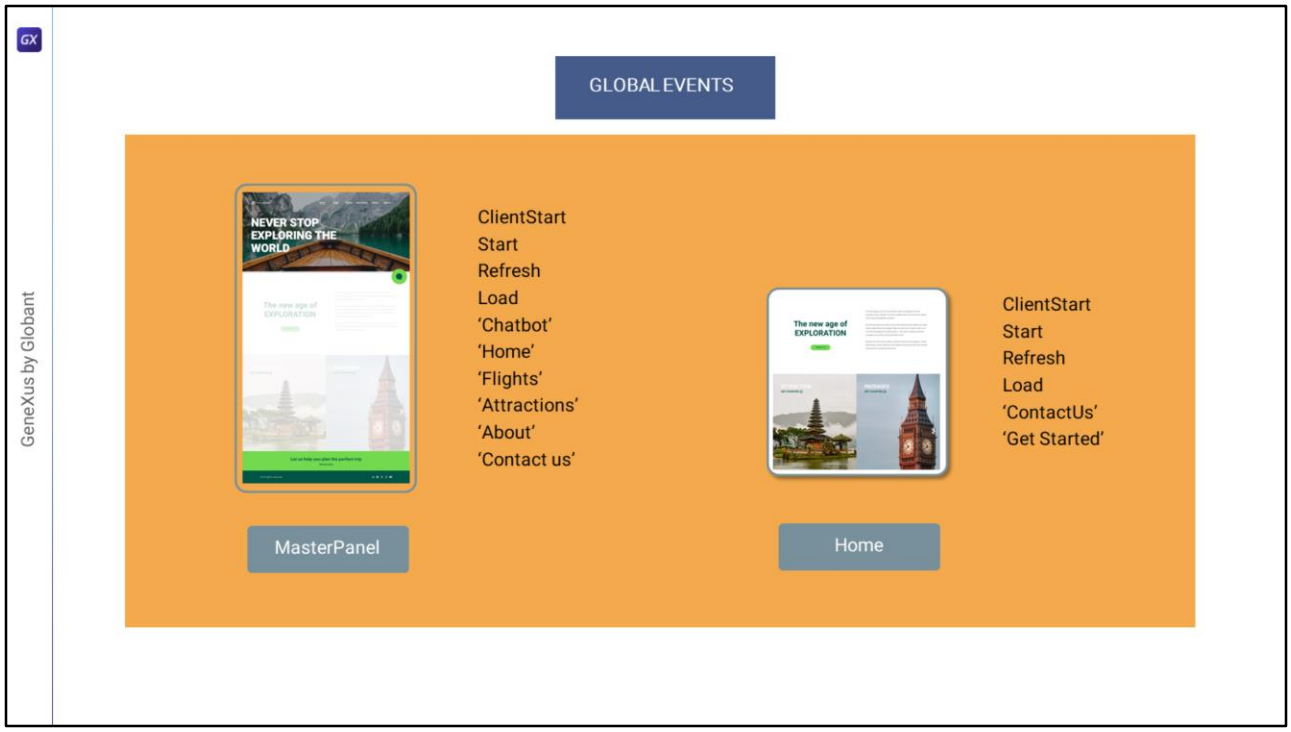
We could think that this is solved by passing parameters, that is to say, each panel identifies itself to the Master Panel in a parameter. But the Master Panel does not support parameter passing.

So, what can we do to achieve this communication between objects that cannot pass parameters? A good option is to use global events.

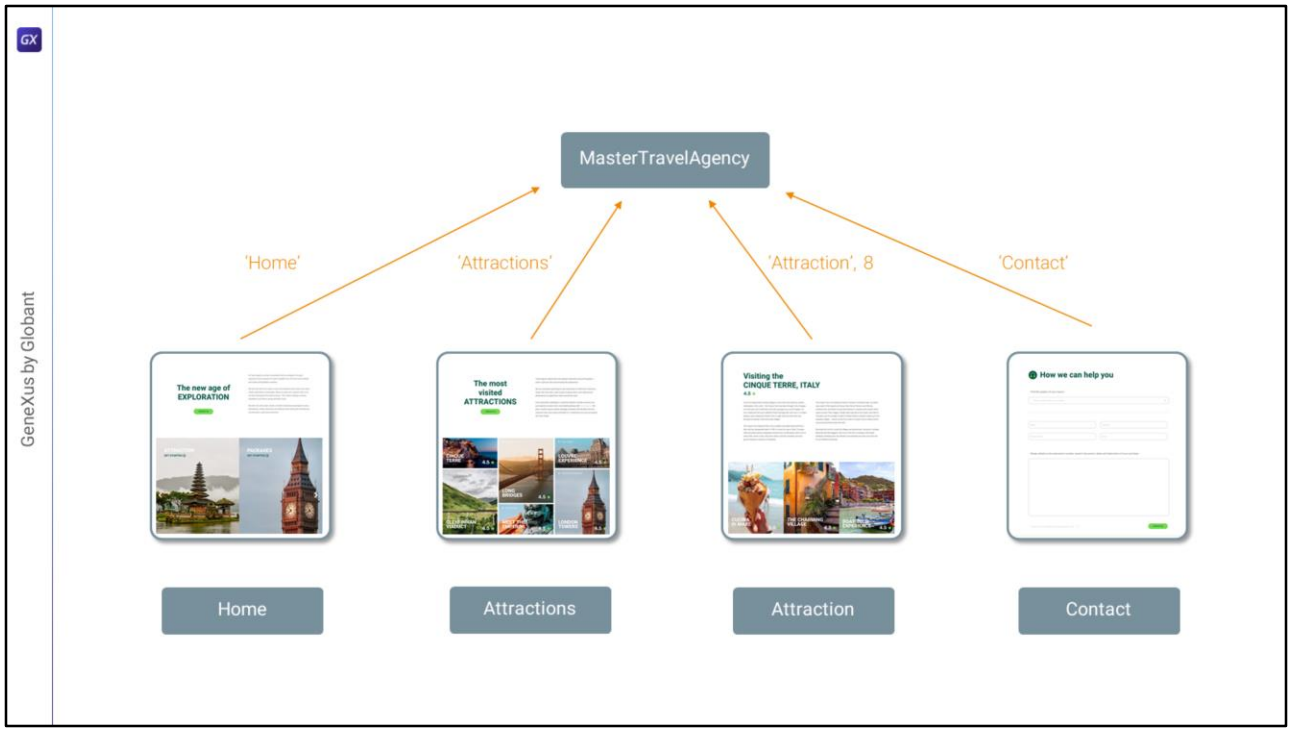


The events we are familiar with so far are the events of each object, internal to the object, both triggered by the user and by the system. For example ClientStart, Start, Refresh, Load, which are system events, or the events associated with the User Interface controls.

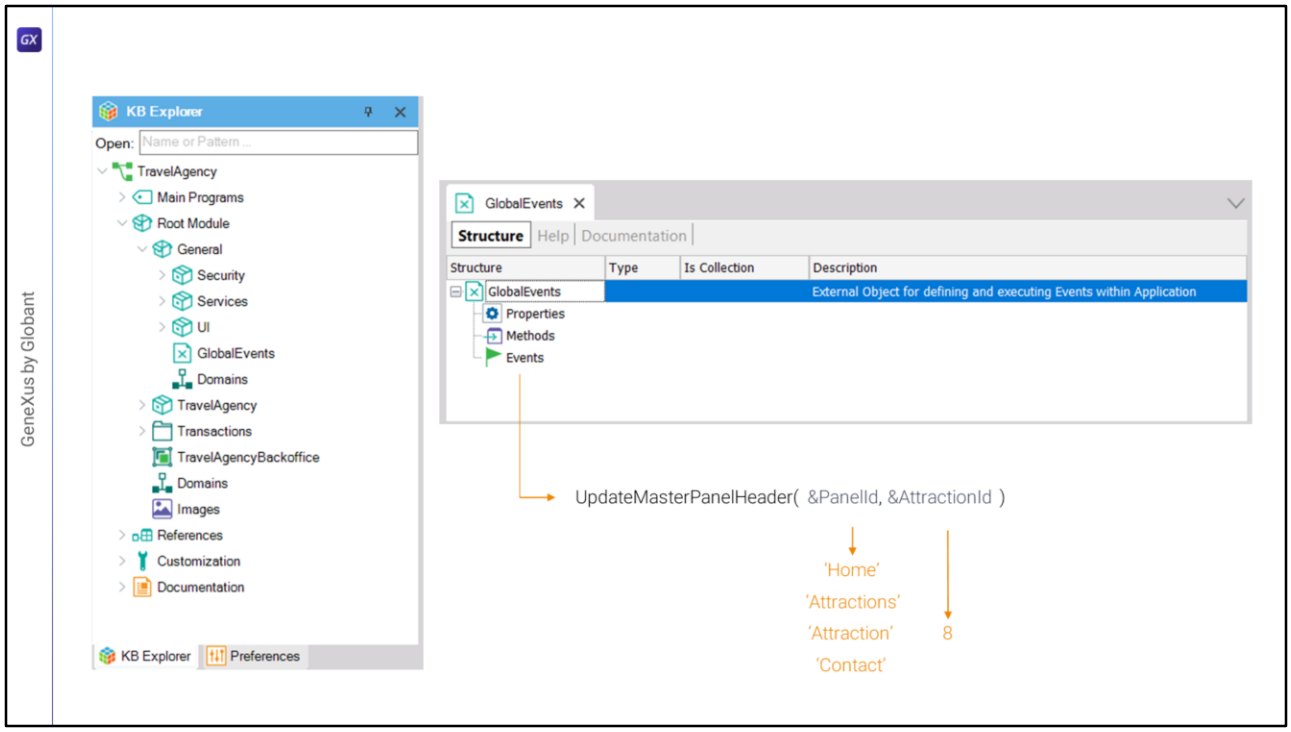
But these are all local events, they are fired and handled within the object that defines them, internally.



On the other hand, **global events** will be valid for all the components of the context of an object that is being executed. In our case, the Panel and the Master Panel are in the same execution context. So, if the Panel triggers a global event, the Master Panel, which is in the same execution context, will be able to listen to it and execute it. The same applies to components, as I will tell you later.



And this is what we need. That the panels trigger an event that is listened to and executed by the Master Panel, through which they can pass information to the Master Panel.

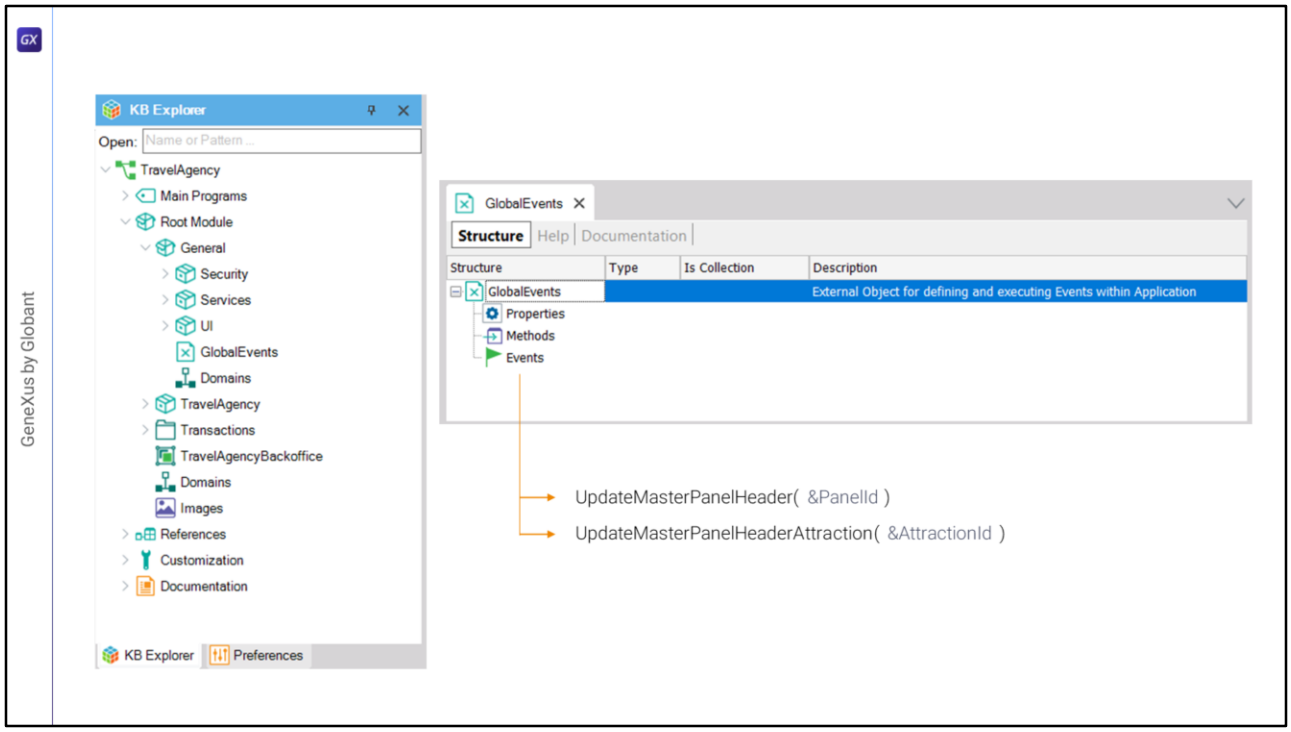


To this end, all KBs provide within the General module a GlobalEvents object, valid for all platforms (web and native).

There, below its Events node, we can define global events, so that they can be shared between objects in the same execution context.

For our case we could define a single event, which when triggered (in our case, from the 4 panels) will send to the listener (in our case, the Master Panel) **2 parameters**: the first one to identify the object that is triggering it (it will be 'Home', 'Attractions', 'Attraction' or 'Contact'), and the second one will only make sense if the one that is triggering it is Attraction, because it is to identify the tourist attraction.

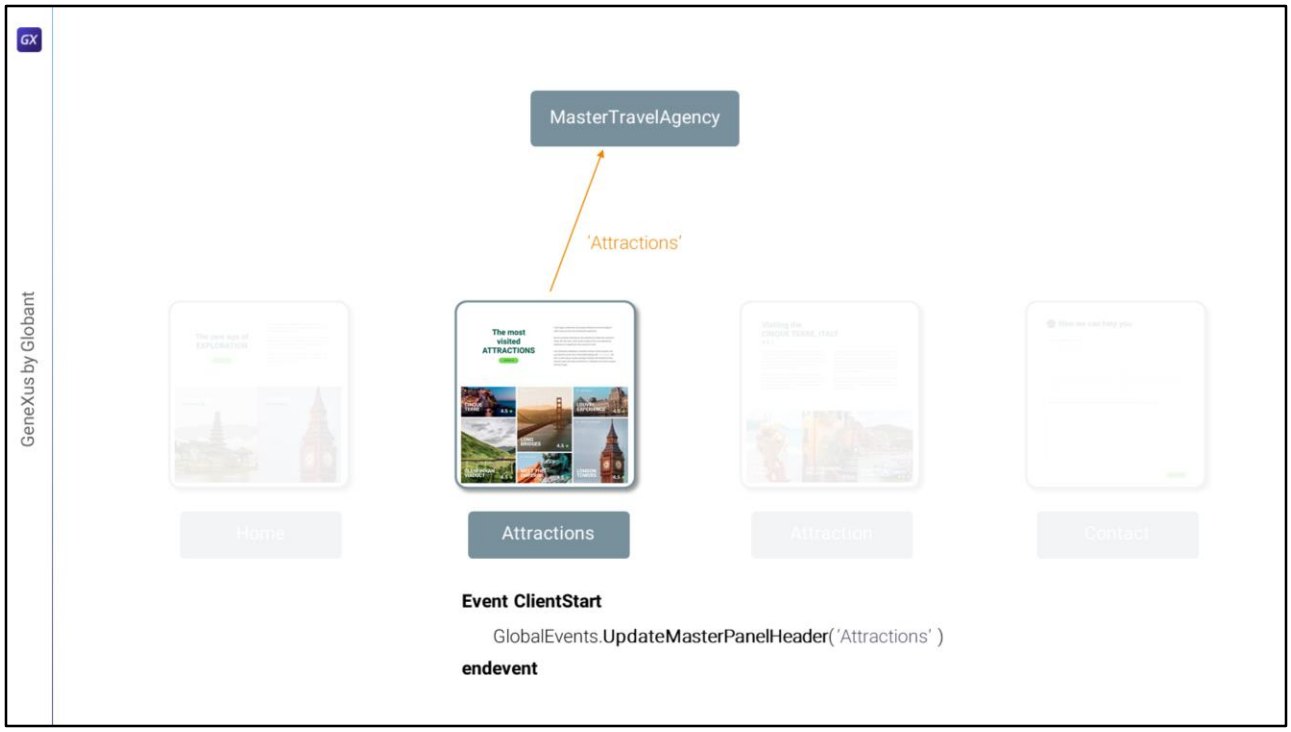
If the triggering ones are Home, Attractions or Contact, the value passed for the second parameter will not be taken into account by the Master Panel.



The other option is to differentiate two events, one that will be invoked from Home, Attractions or Contact for identification, and the other that will be triggered only from Attraction, sending the attraction ID.

Both solutions (a single event, versus two distinct ones) have pros and cons. I'm going to choose the second one, with two events, just so you can see that we can add all the global events we want to this GlobalEvents object.

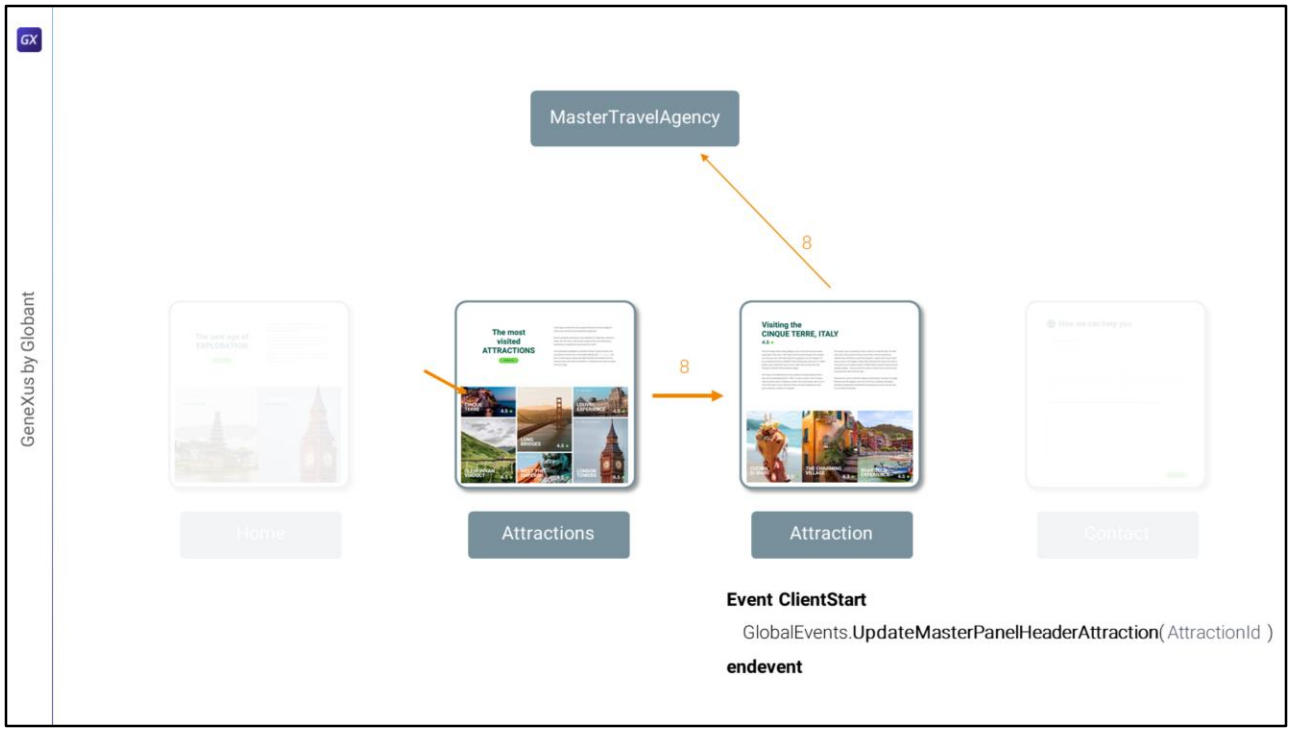
Then for the Home panel, in the ClientStart I will fire the UpdateMasterPanelHeader event, passing the value 'Home' as parameter.



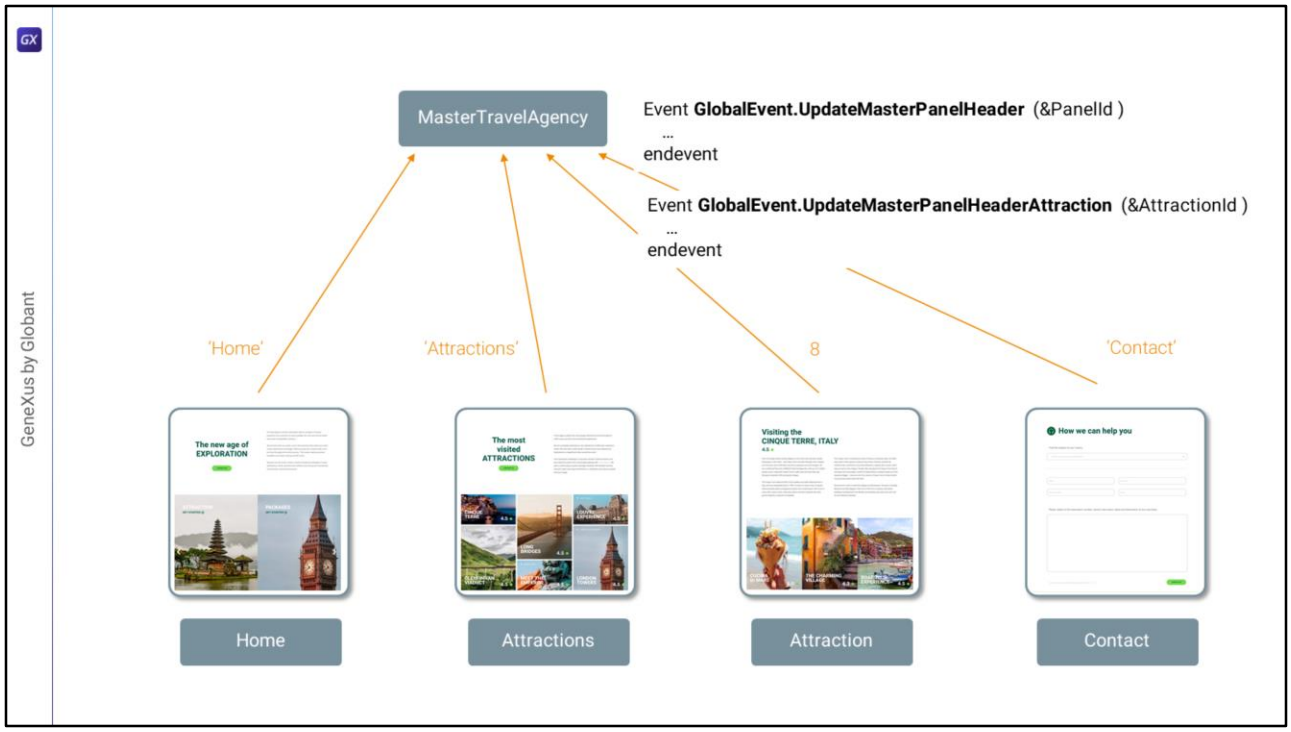
And the same will be done in the ClientStart of Attractions, but passing, of course, 'Attractions' as a parameter...

...and in Contact, but now passing 'Contact'.

In that of Attraction the other event will be fired; it will be the particular one, UpdateMasterPanelHeaderAttraction…

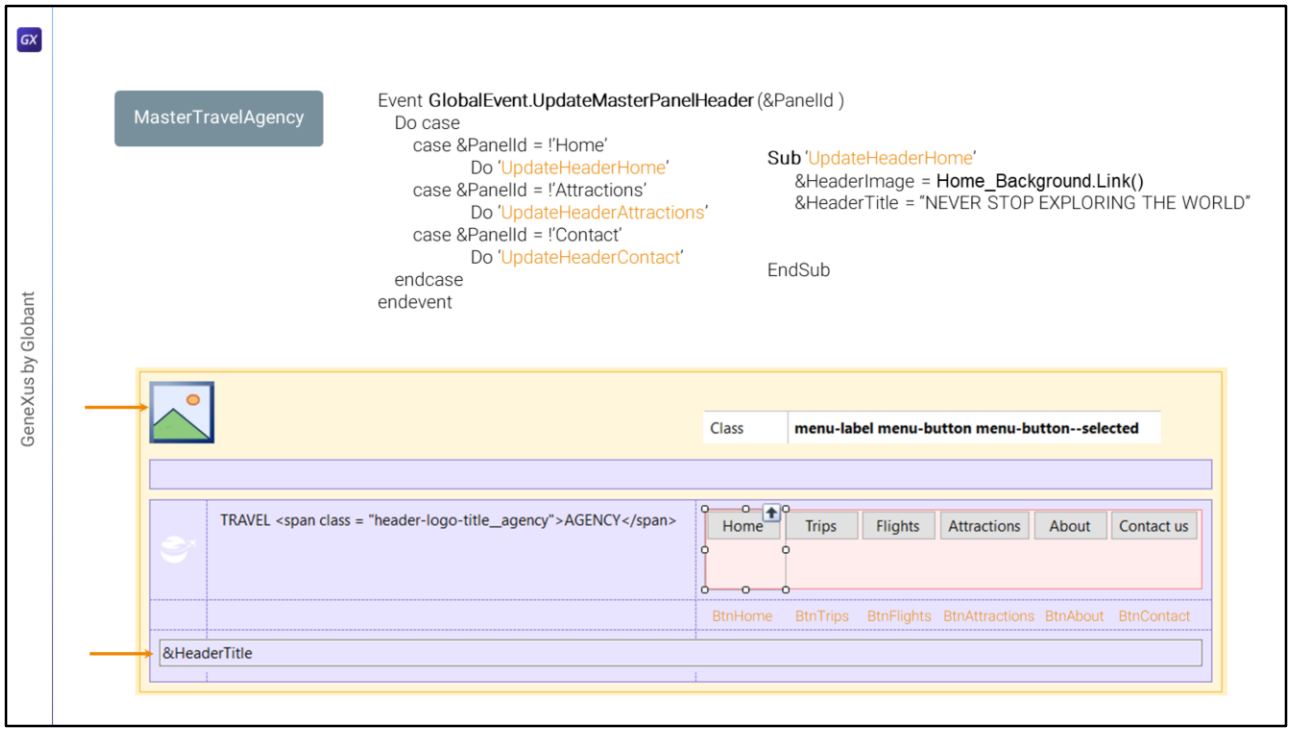


And since the Attraction panel will receive by parameter the ID of the tourist attraction (which will be sent from here, when the user selects an attraction to see its information, for example, the one with ID 8), this same attraction ID will be sent by parameter to this global event.



Then, the Master Panel will only have to “listen” to these two events, to receive that identification and proceed.

In this way, if any of these three panels are running, they will trigger this event.

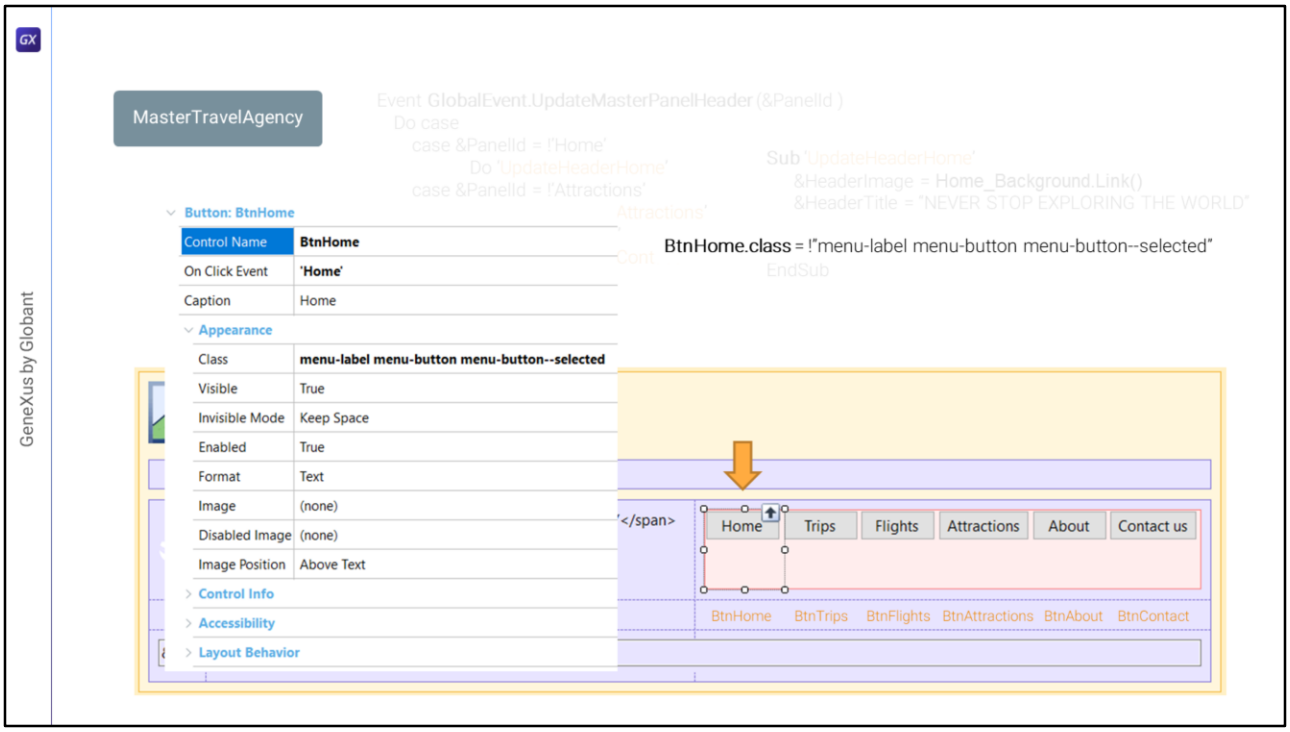


And what does the event have to do? Based on what fired it... (this exclamation mark before these texts is to instruct GeneXus that these texts are internal to the application and, if the application is translated to other languages, these texts should not be translated)... Well, as I was saying, based on what fired it, it should update the Header by loading the two variables: &HeaderImage and &HeaderTitle with the values that correspond to that panel that fired it... and for the buttons, it will have to leave them all cleared except one, the one that corresponds to the panel.

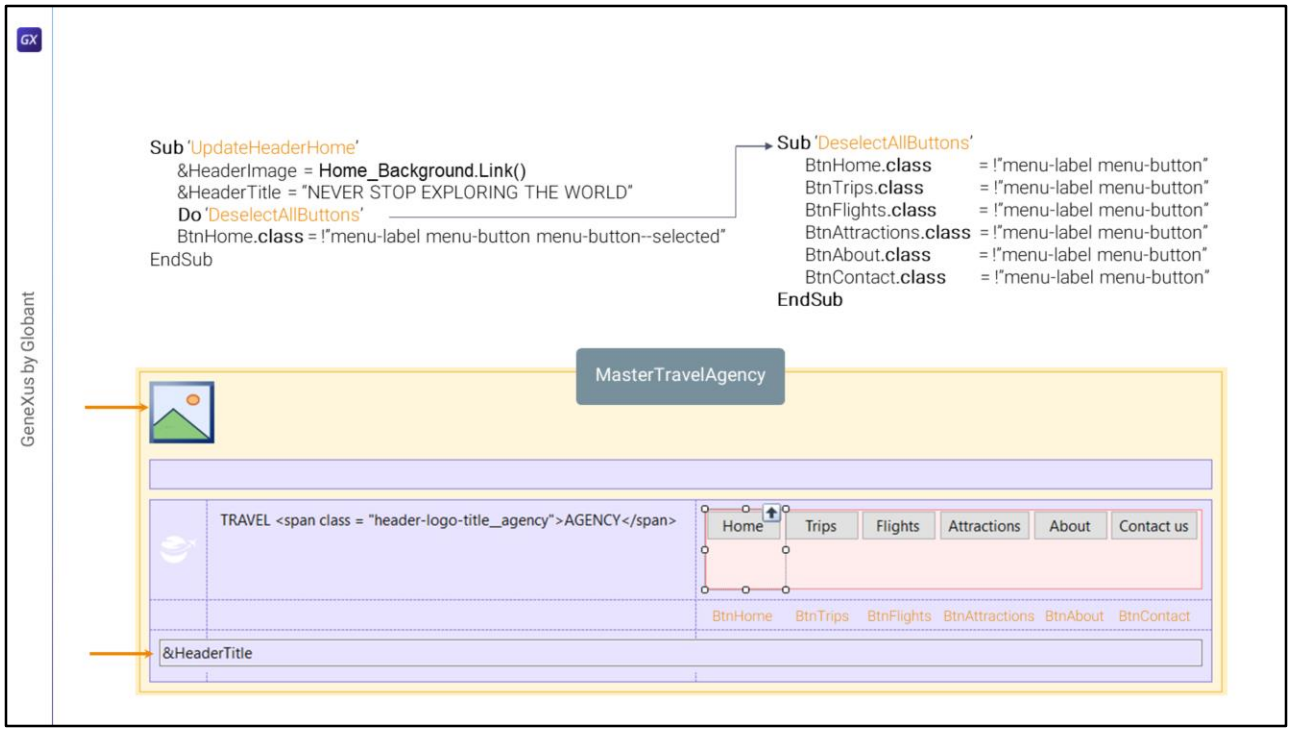
What we are writing here are invocations to subroutines, which are portions of code local to the object, which can be isolated to reuse them or to make the code semantics more understandable.

Within each of these subroutines, for example, let's look at the Home one... the image variable and the title variable will be updated... and the menu will be left with the correct option selected. How do we do the latter?

If we name the button controls in this way... remember from the previous video that we had two classes for all the buttons: menu-label for the typography and menu-button to make the bottom and top border transparent, remember? (we could have joined both into one, but well, we didn't do it). There is a third class to achieve the green indicator below, which was “menu-button--selected”; in the previous video, we had left it associated in a static way to the Home button.

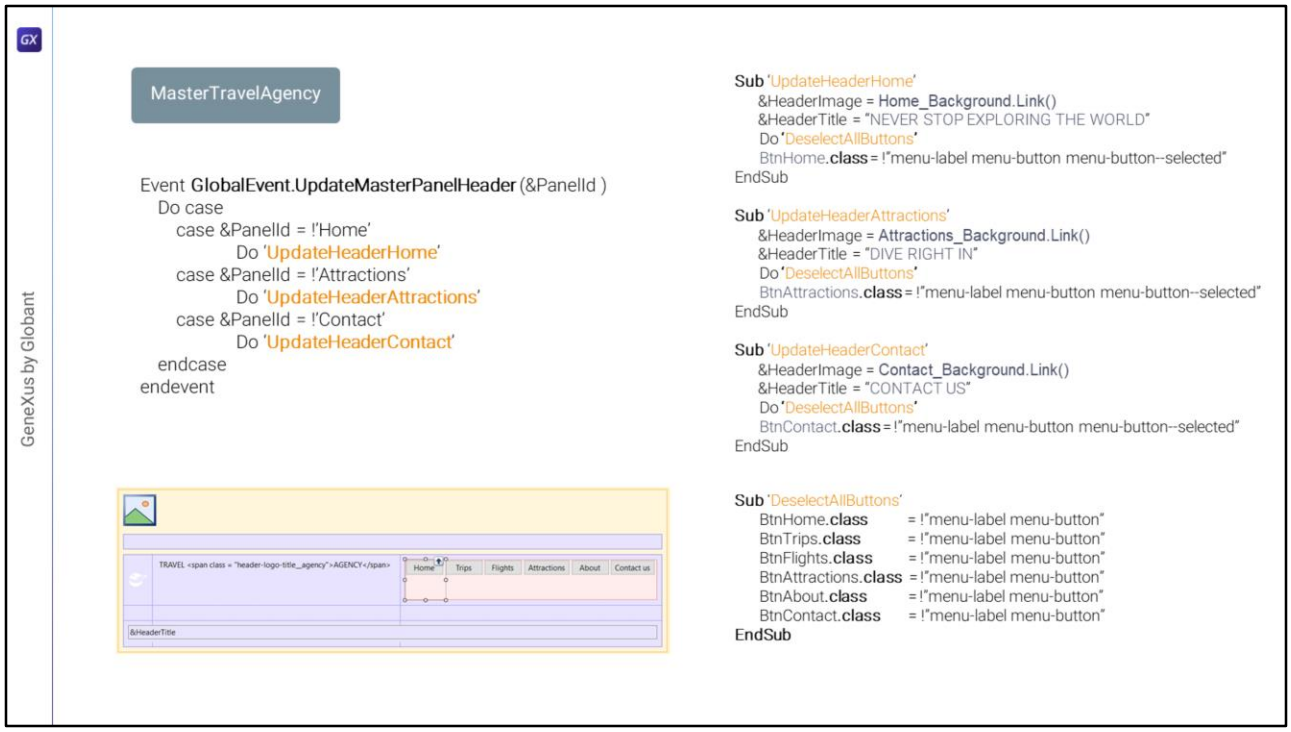


Classes can be assigned to controls both statically, through the control property, and dynamically, anywhere code is accepted, such as in events.



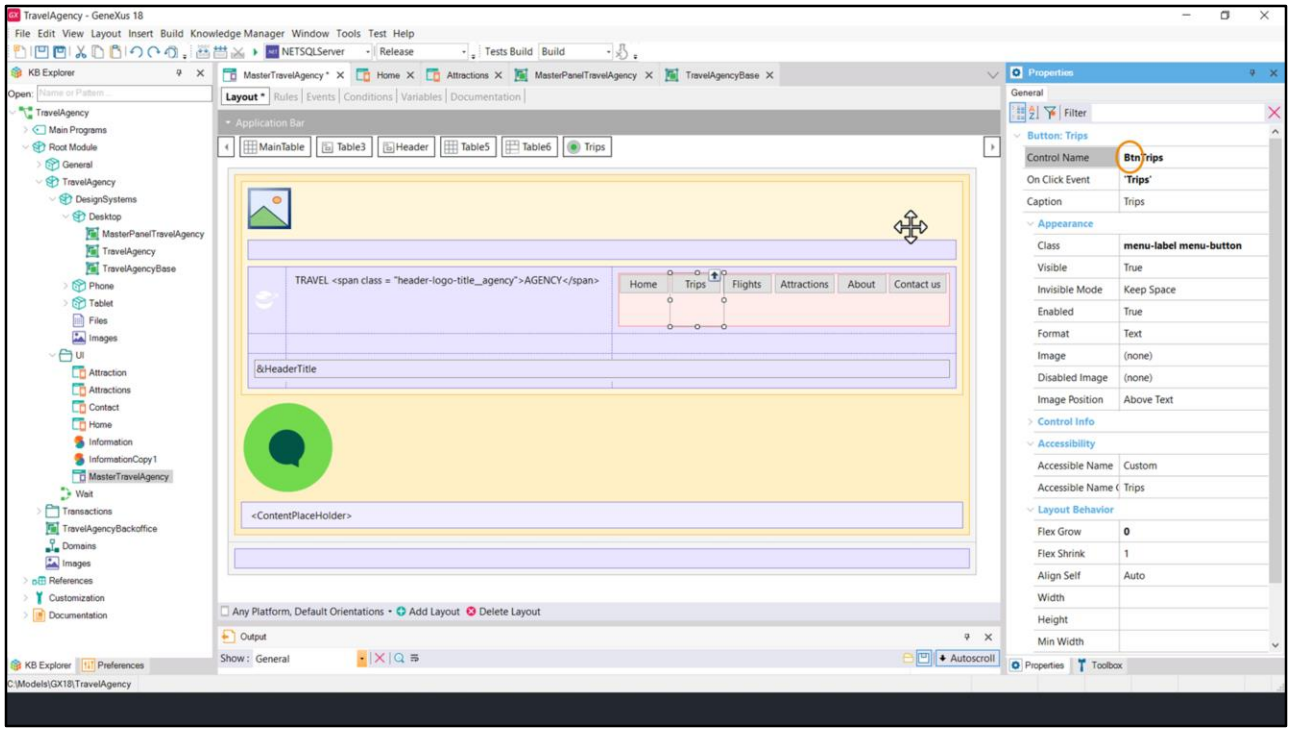
In this way, we could write a “DeselectAllButtons” subroutine that, as we can see, assigns the two classes, menu-label and menu-button, to all the buttons of the menu.

Then in this subroutine from which we start we would first invoke this other subroutine to leave all the buttons as deselected. And then simply assign to the corresponding button, which in this case is the one named BtnHome, its classes, which are the same as those of the others, plus menu-button--selected.



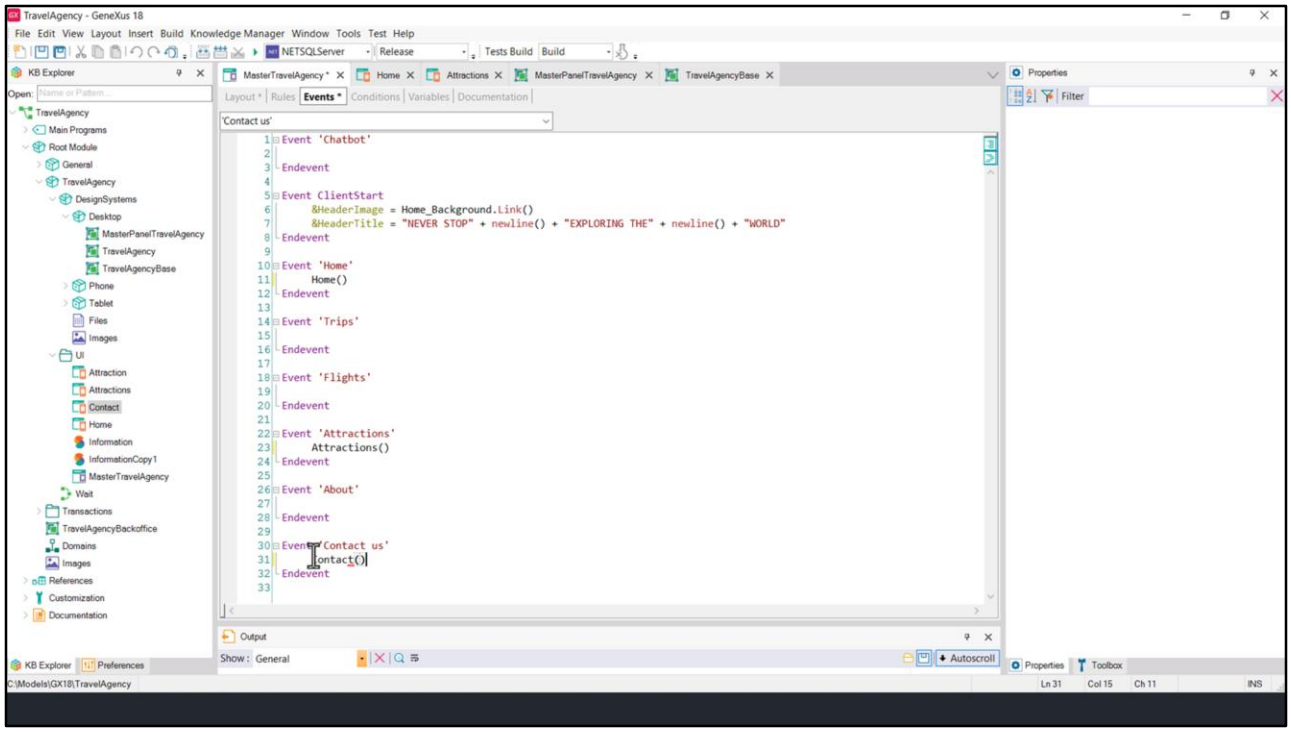
This is how the events tab of the Master Panel will look like in order to correctly implement the Header when the Master Panel is called by the Home, Attractions or Contact panels. Note that the subroutines for the last two have exactly the same logic that we saw for Home: they load the corresponding image, the title, and clear all the buttons, and then leave only the corresponding one selected.

Now we need to implement the correct loading of the Header when the caller is Attraction, but first let's take all this to GeneXus, to be on the safe side.

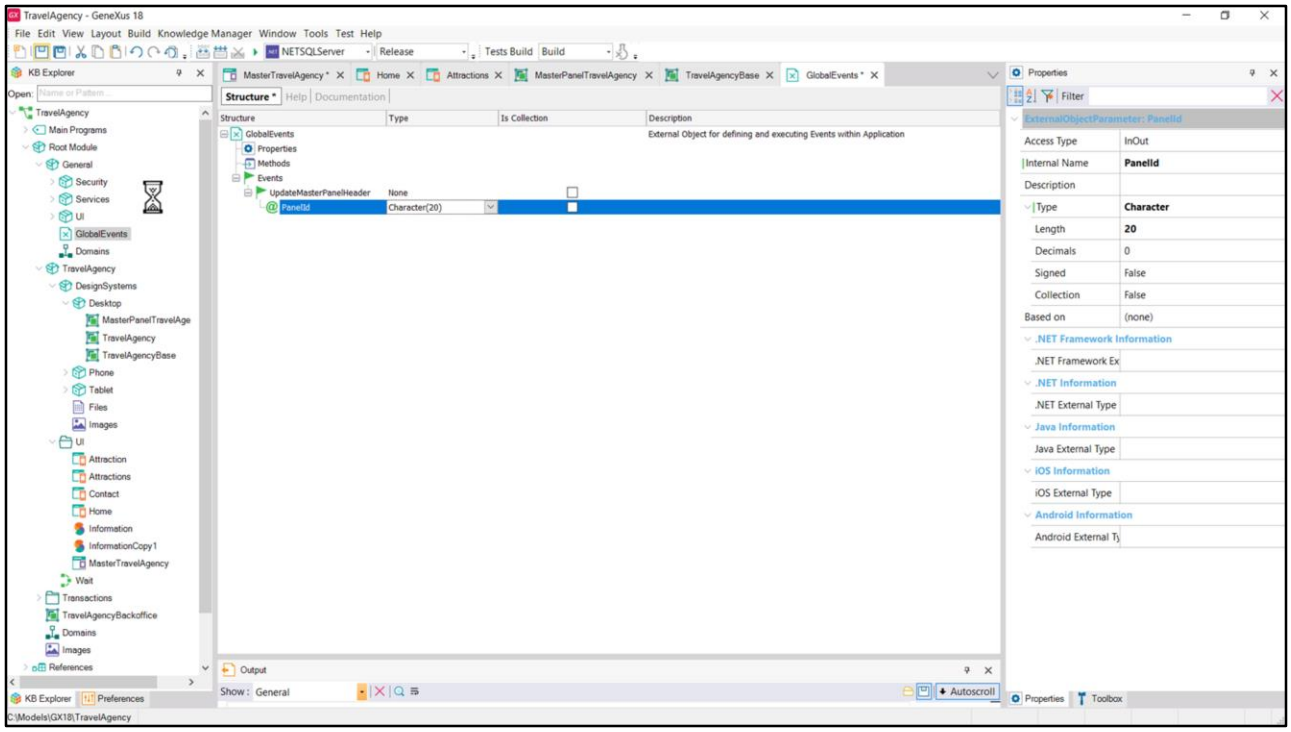


Here we have the Master Panel. Remember that due to a bug in the editor, the buttons were invisible. For the moment, to make them visible again and to be able to operate with them, let's modify this property, which was the one related to the bug. Later we will leave it as it should be, with the value Center.

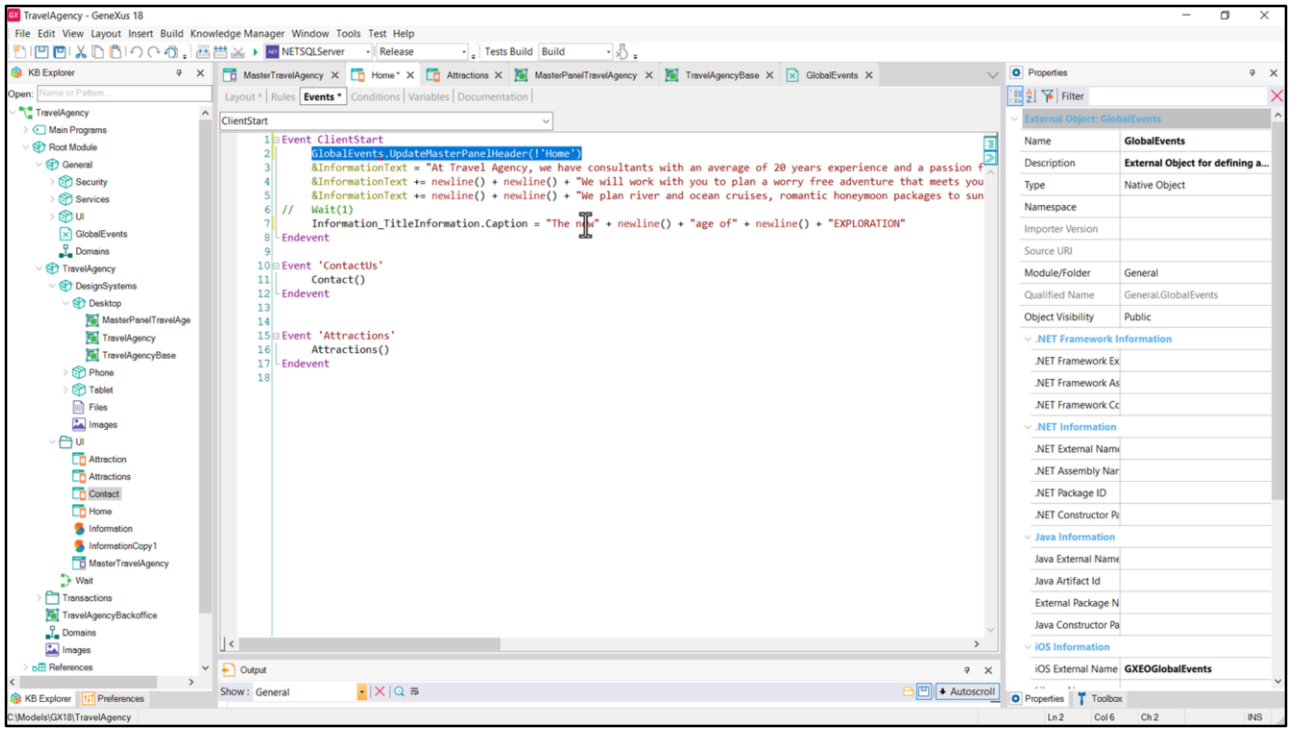
Let's take this opportunity and change the name of the buttons to precede them with “Btn”, so their use will be clearer later in the code.



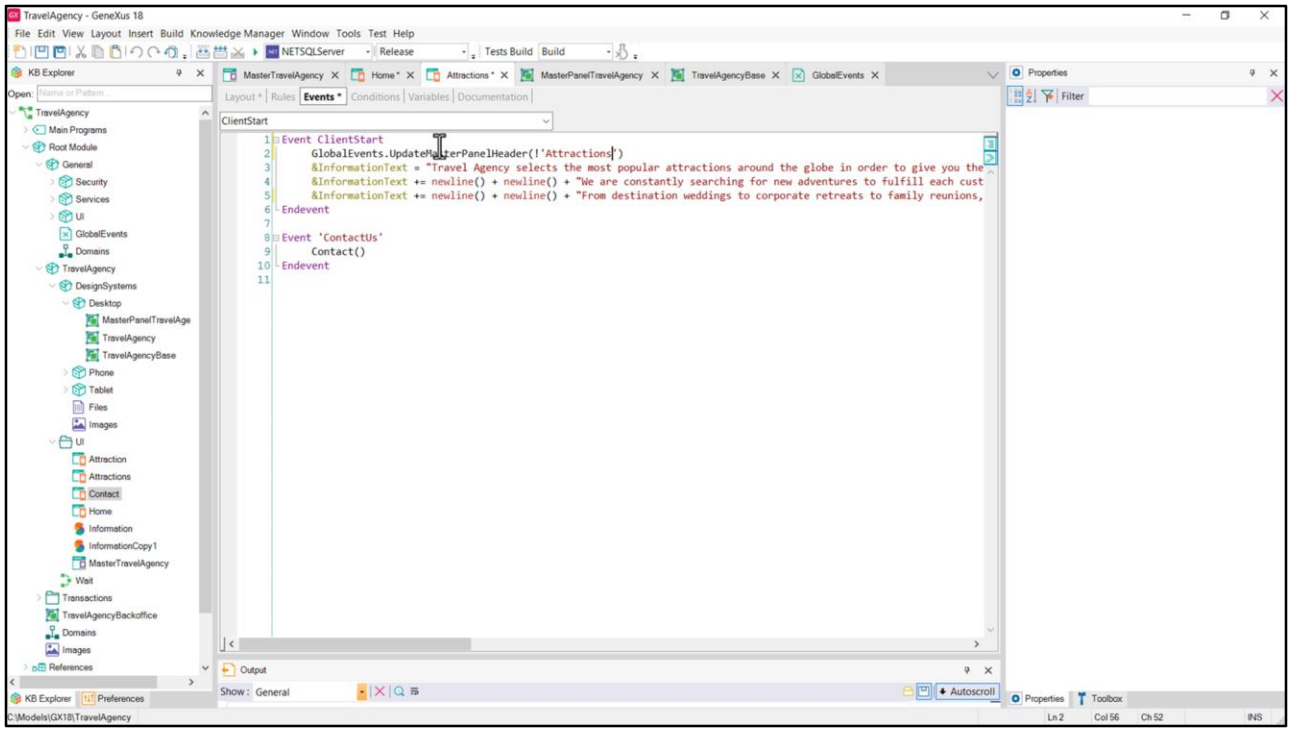
Let's start by making the invocations to the panels in each button. In this way, when the Home button is clicked on, we will invoke the Home panel. When Attractions is clicked on, we will invoke the Attractions panel and when Contact us is clicked on, we will invoke the Contact panel. For the moment we haven't even designed the other panels.



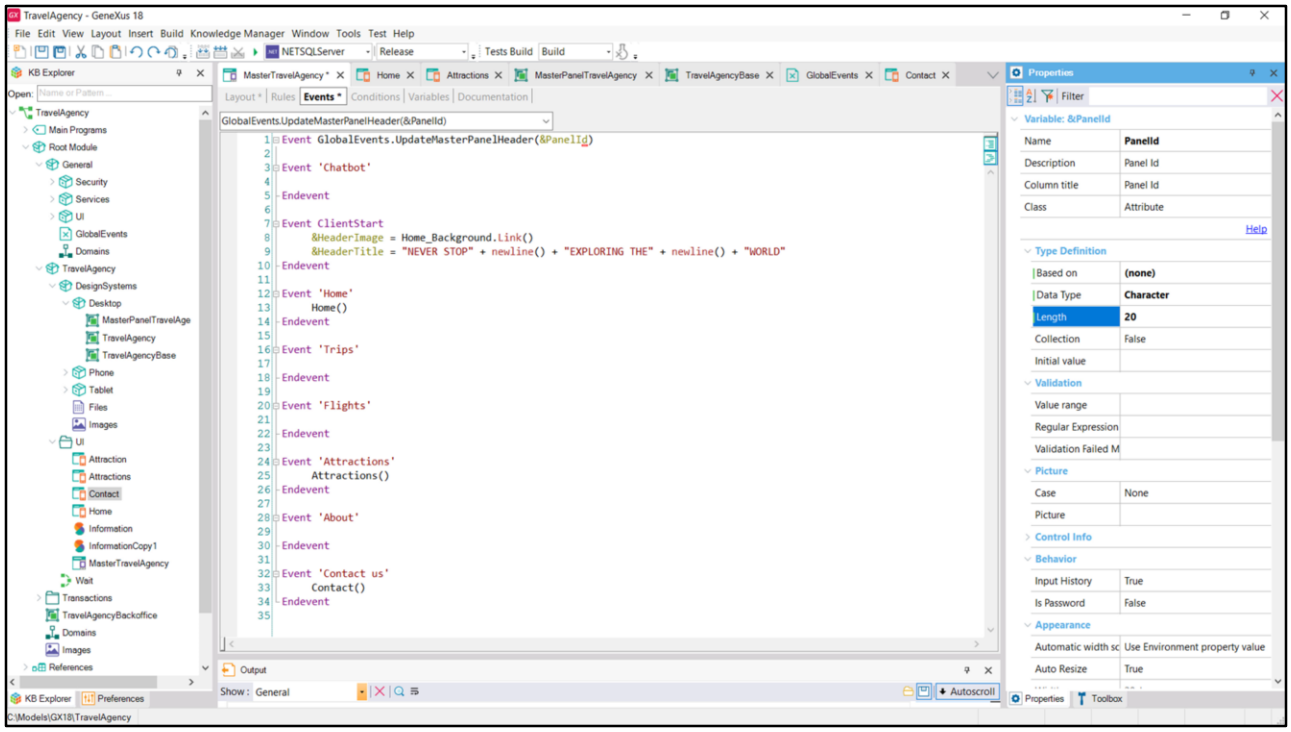
Now let's edit the GlobalEvents object. We will add the UpdateMasterPanelHeader event with the variable &PanelId of Character type.



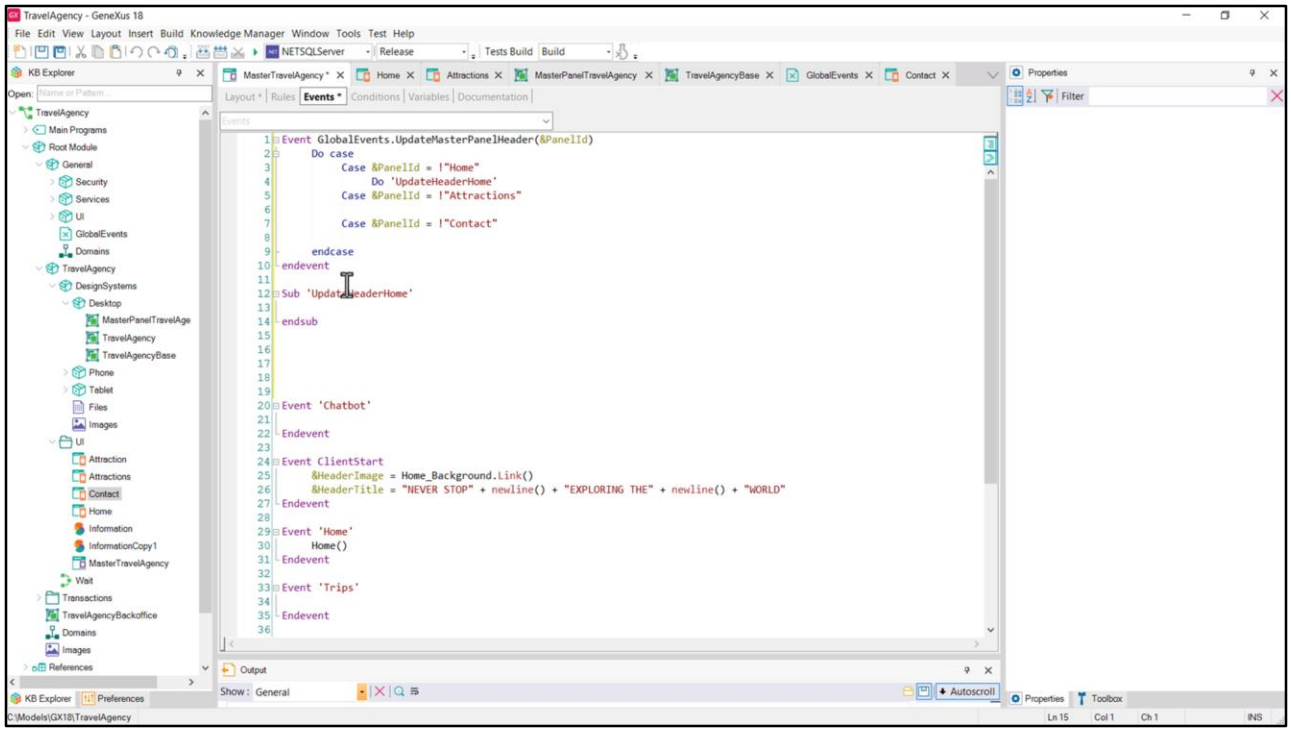
We go to the Home and fire it in ClientStart. I'll place it at the beginning of the code to make it clearer.



And we do the same in Attractions, changing here for 'Attractions'. And in Contact.

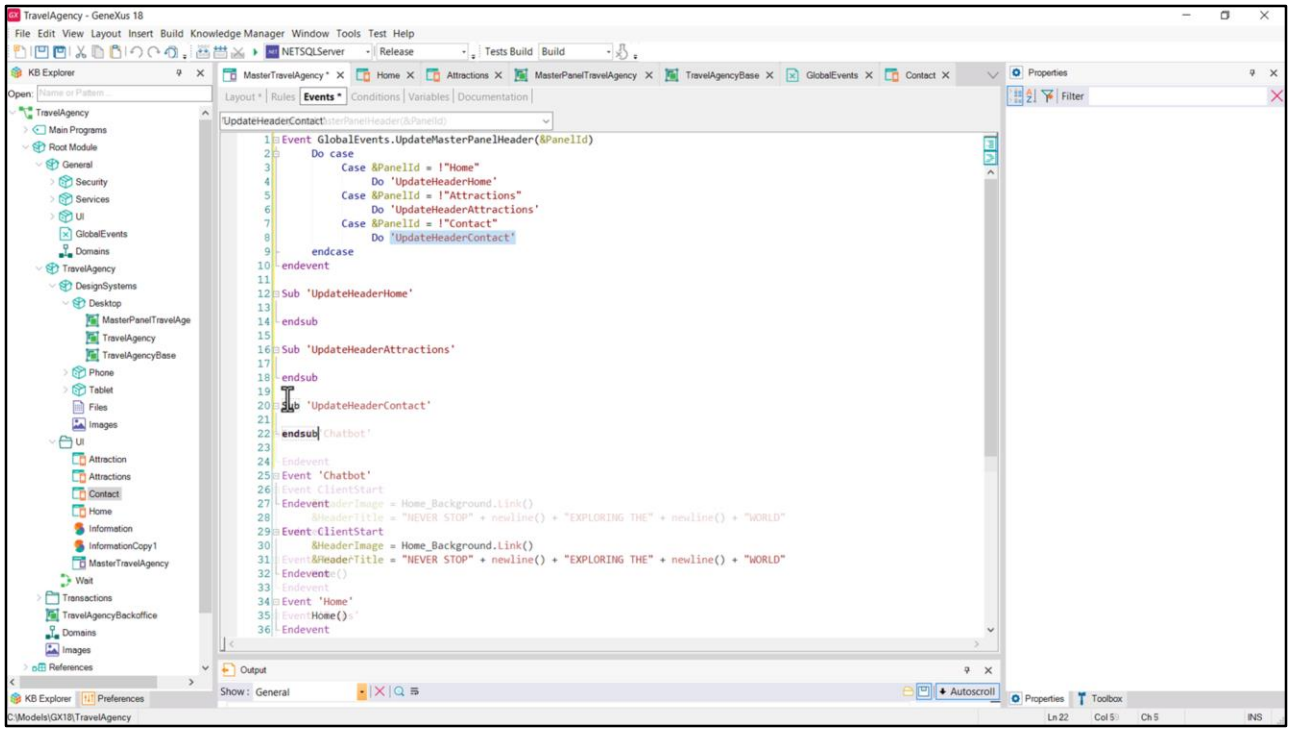


Now we will program the global event in the Master Panel... Let's define the variable... &PanelId... we don't want it based on the Id domain but we want it to be of Character type.

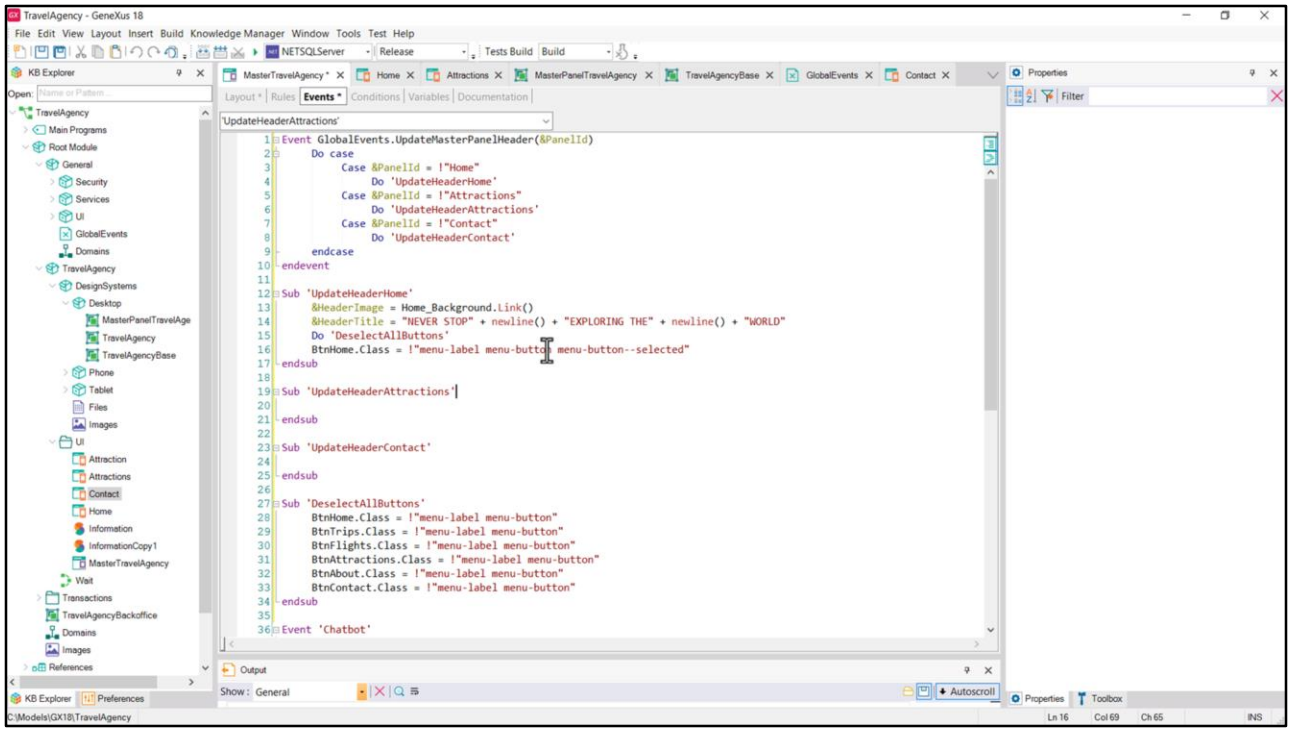


Well, we add the **Do case** command in order to take different actions based on the value of the variable... If it is 'Home' one thing will be done, if it is 'Attractions' another, if it is 'Contact' something else.

What will be done if it is 'Home'? Invoke this subroutine, which we have to define... We can specify it anywhere in this events tab, which as we know is declarative; the order in which we define the events and the subroutines does not matter.

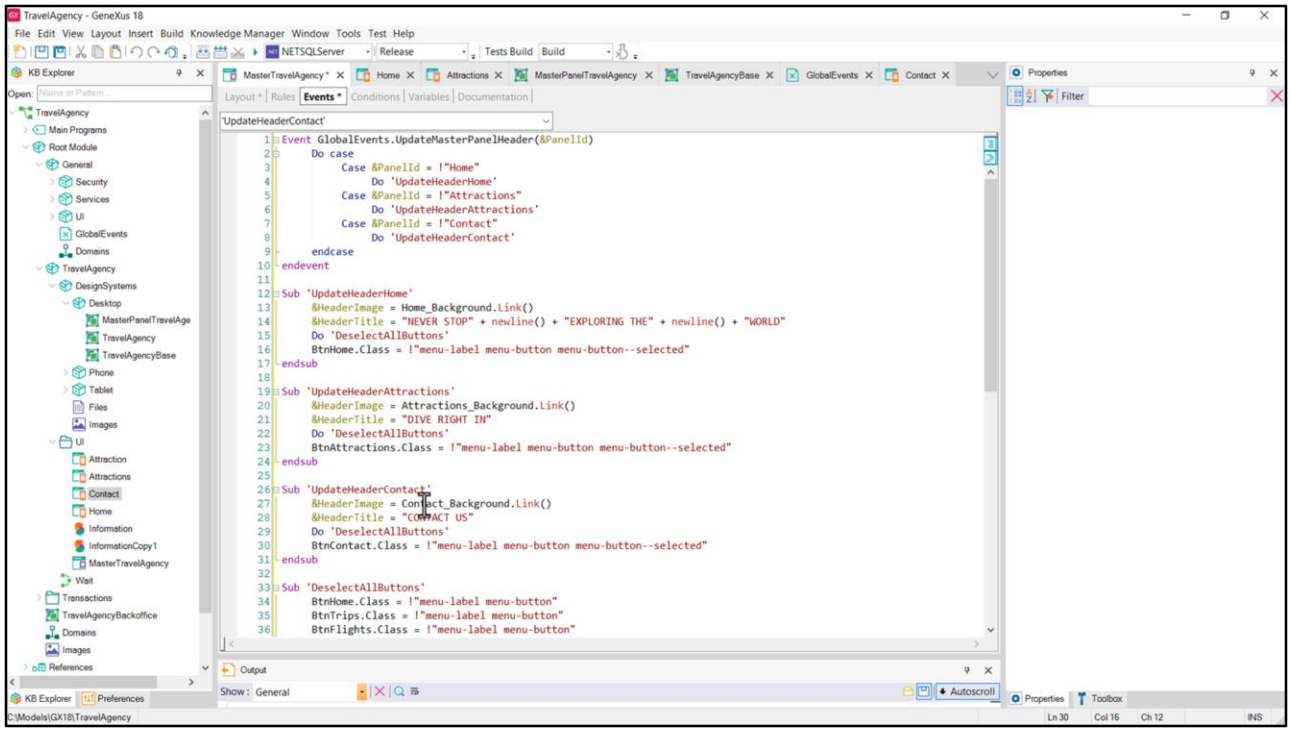


If the value taken by the variable is 'Attractions' we will call this other subroutine. And if it is 'Contact', this other one.



Let's program the first one. To the &HomeImage variable we assign the image that we had inserted in the KB in the preparation stage, link method.
To the title this one here.
Then we invoke the DeselectAllButtons subroutine that we will define later. And finally to the Home button we assign the 3 classes.

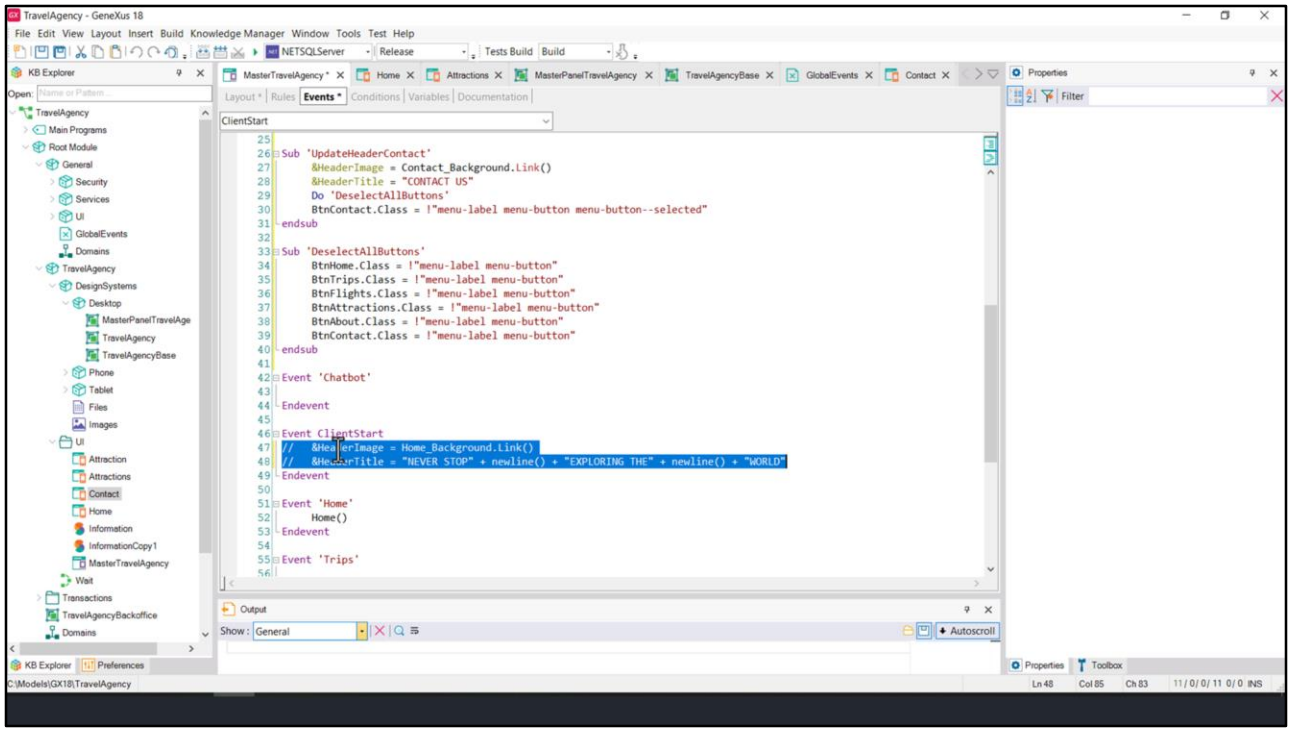
Now let's specify the DeselectAllButtons subroutine, in which we assign the first two classes to all the buttons of the menu.



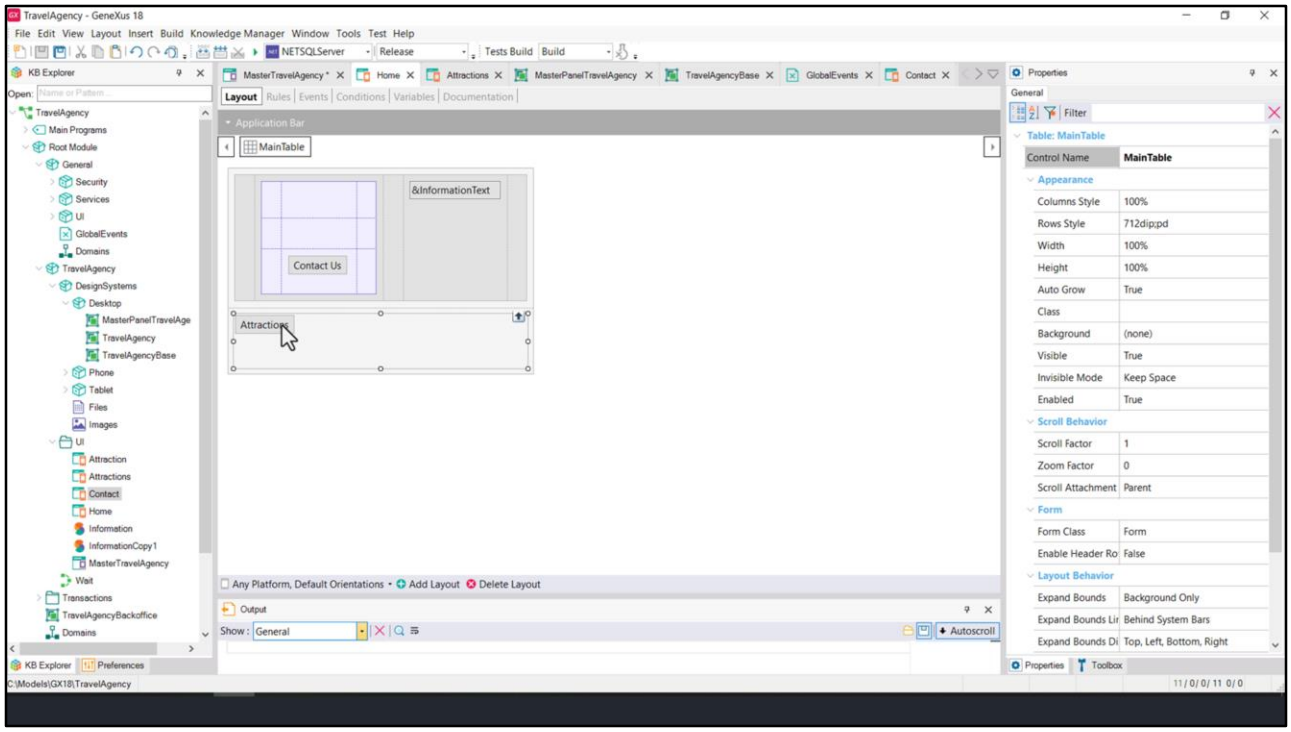
Similarly, we program the other two subroutines....
Here the image was called like this... and the text was this...(we extracted it from Figma)... and the button here is the Attractions button.

And for Contact we also make the necessary replacements.

When saving it gives an error... it doesn't understand the name BtnContact. We come here... it was called this way. We will remove this ending. Now it works.



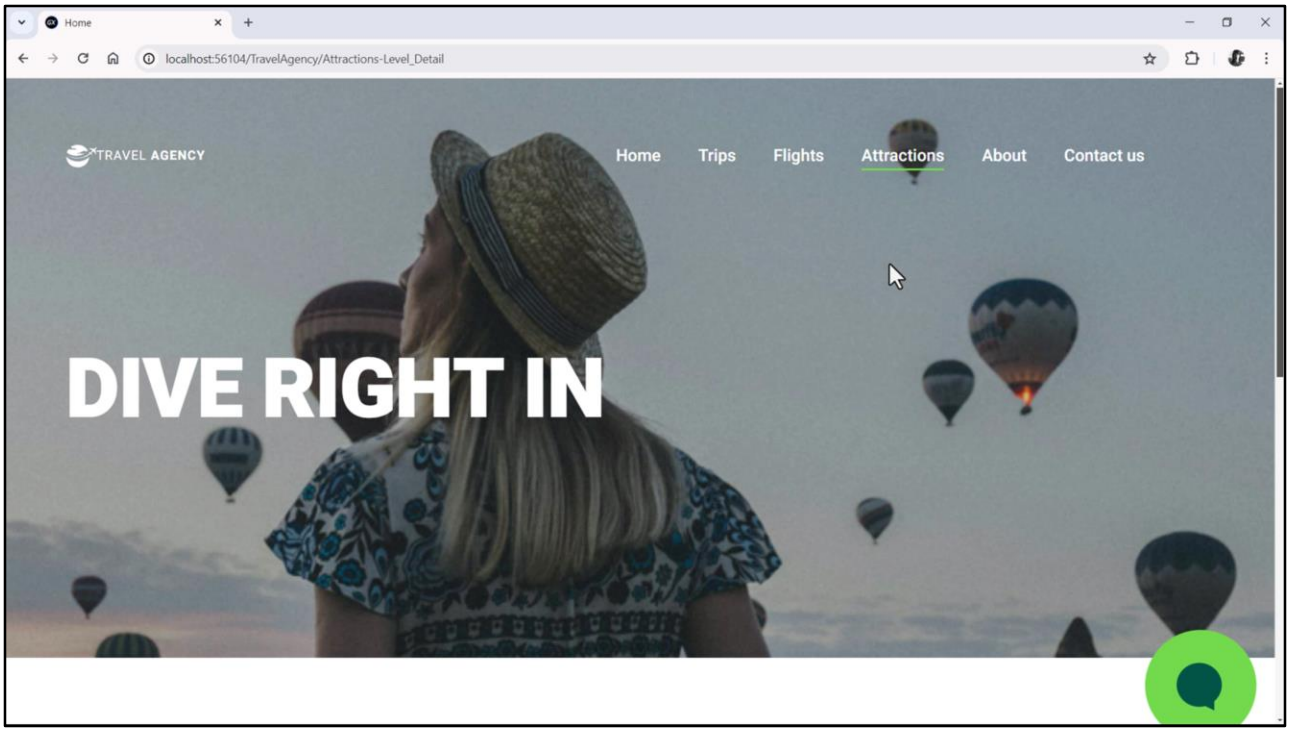
This loading of the Master Panel ClientStart will no longer be necessary.



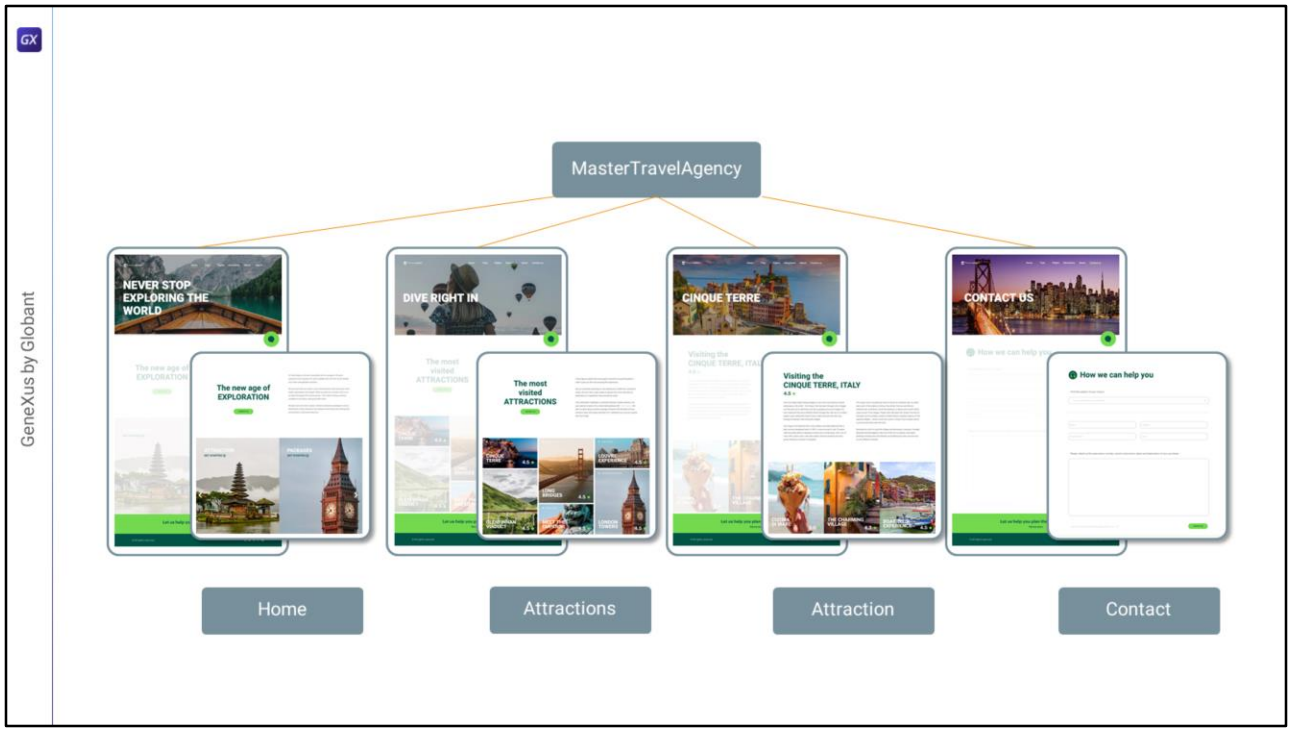
And neither will be this button to call the attractions panel.

We check everything here... it's all right.

Let's return the Align Items property of the flex to Center. And run it.

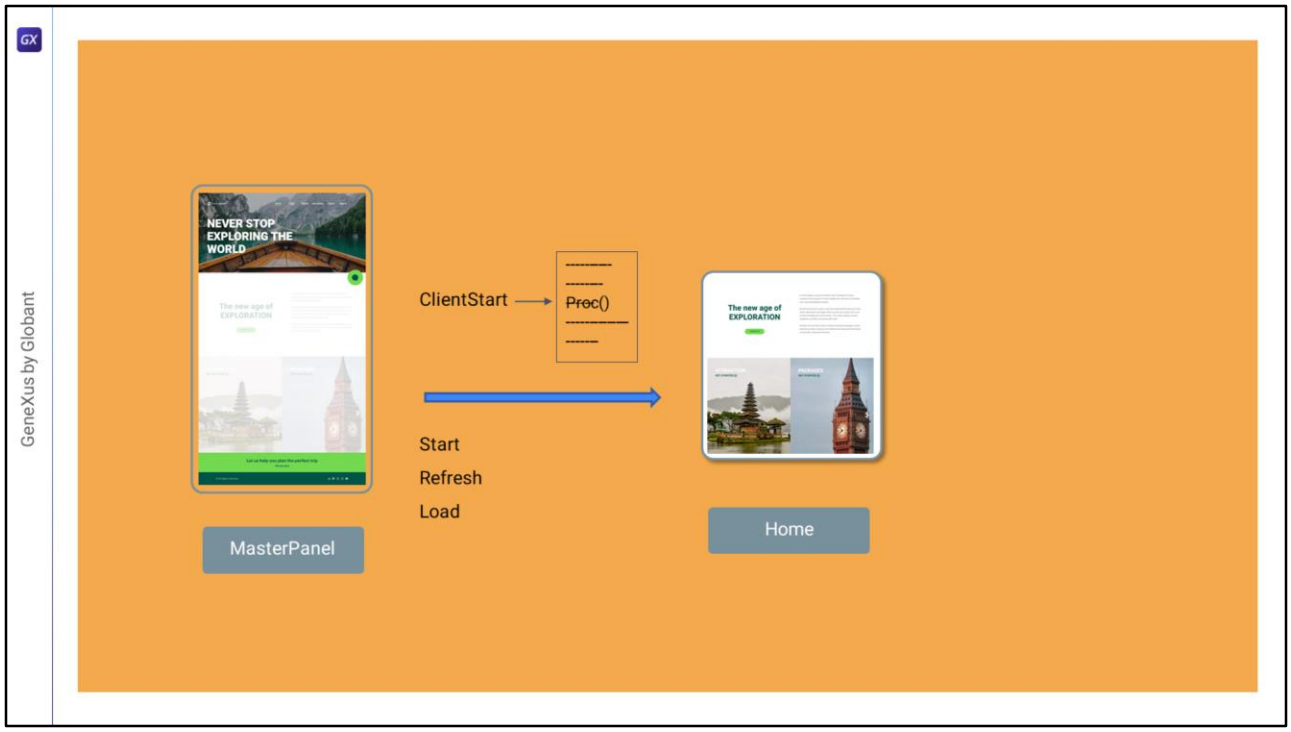


The Home page looks perfect... and if we click here, the attractions page is also perfect, and the contact page too. We go back to Home... everything is perfect.



At the moment (and this is being modified, so whether this is still the case or not will depend on when you watch this video), every time a panel is executed its Master Panel is executed again, regardless if the Master Panel is the same between panel executions. That is, it is built and destroyed between navigations, every time.

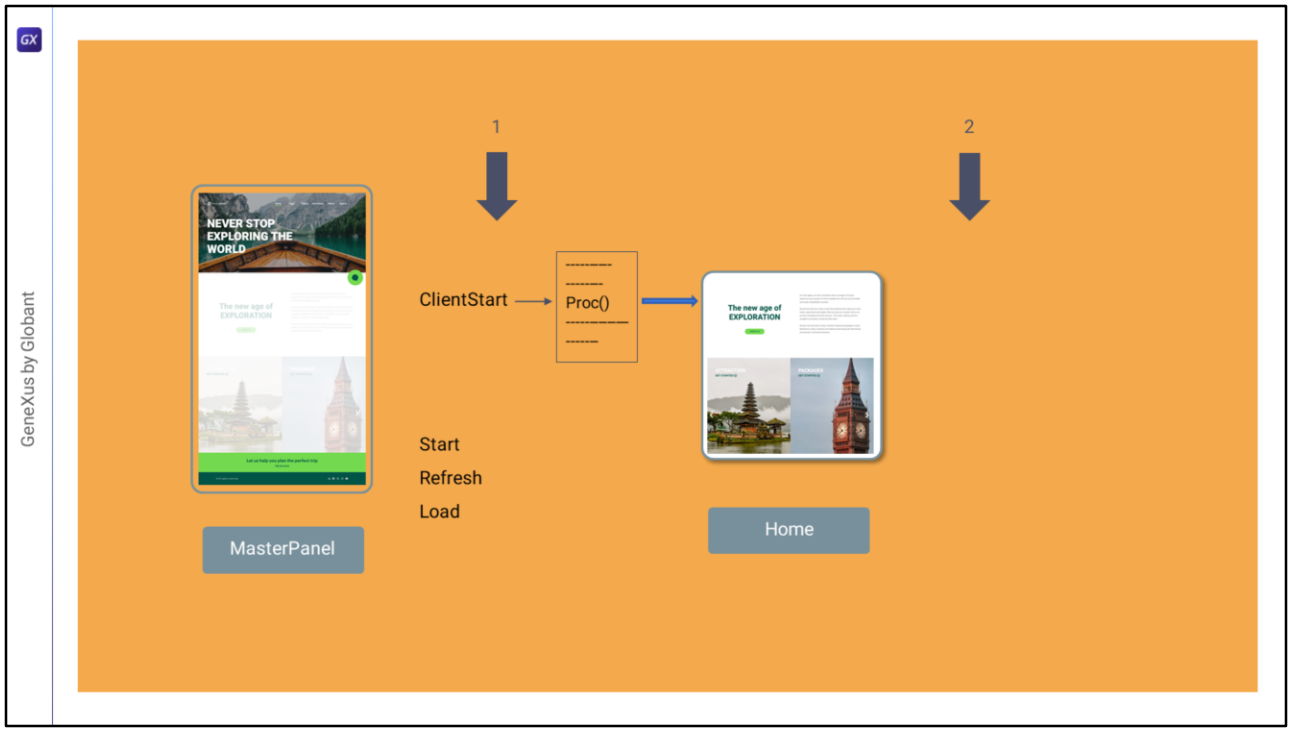
That is certainly not the most efficient thing in the world and can lead to flickering issues, for example.



To understand all this a little better, I'll tell you how it works right now.

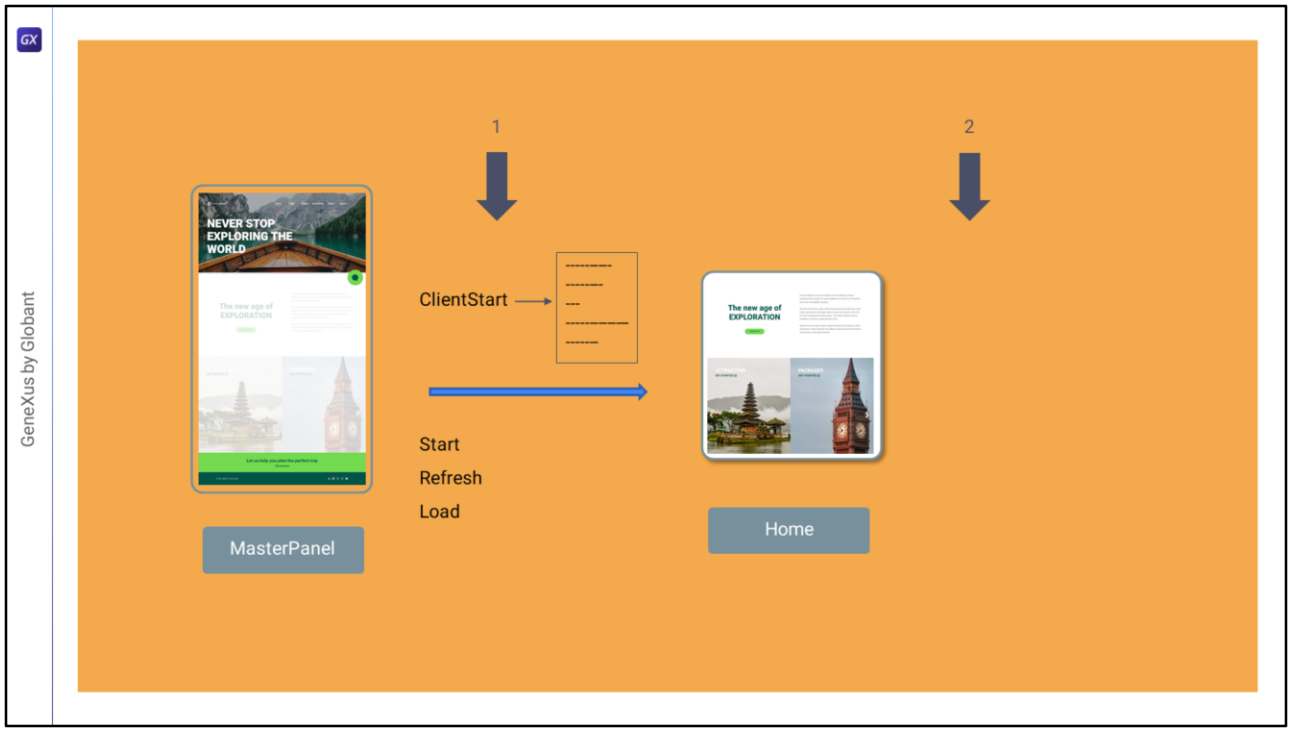
When launching the execution of the Home panel, for example, the order of execution will be as follows:
First the Master Panel is built, where all the static properties of the layout controls are assigned (the ones that were indicated in due time in the properties window of each control; they are the ones we usually call design-time properties), and next the ClientStart event is executed.
If the ClientStart event has an invocation to a procedure, for example, which must necessarily be resolved in the server, or to a Data Provider, its execution is triggered asynchronously, and the Master Panel is rendered for the first time with what it has so far.
That is to say, it doesn't wait for this Proc to end to continue and render only at the end of the event. It is rendered and then it continues.
On the other hand, if inside ClientStart there is no asynchronous invocation, then it is rendered for the first time only when ClientStart ends.
In any case, after the first rendering of the Master Panel, the build of the projected Panel in the ContentPlaceHolder is already started, that is, BEFORE the Start, Refresh and Load events are executed.

Either because we are here, or because we are here.



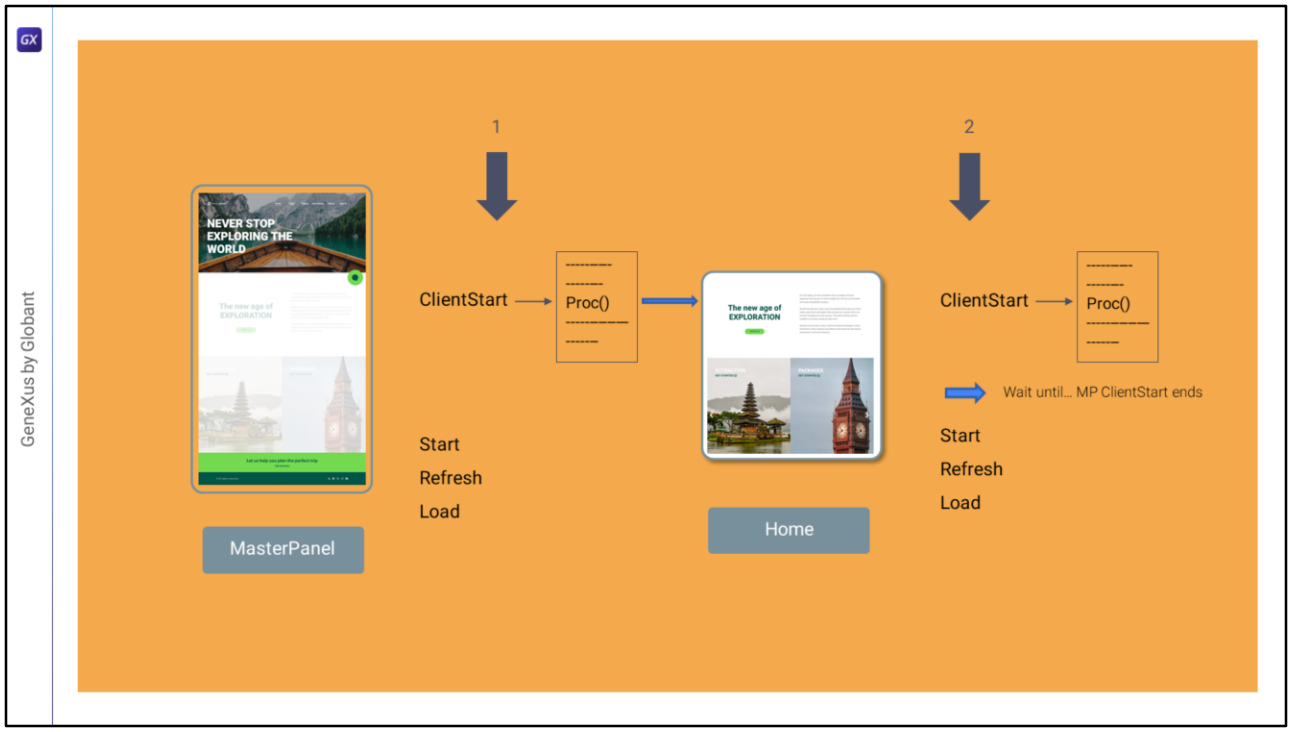
Another concurrent execution line opens.

In the one we were in, the Master Panel was already rendered for the first time and the build of the Panel was launched. When the Proc ends, the execution of the rest of the ClientStart continues, and if in the Proc or here there were changes in the User Interface, it is rendered again. Next, the other events of the system are executed (of course, rendering if they introduce changes in the UI).



And if there was no Proc or anything asynchronous, then here the Master Panel had already been rendered and the Panel build had been launched, so now the other system events are executed.

What happens in the concurrent line?

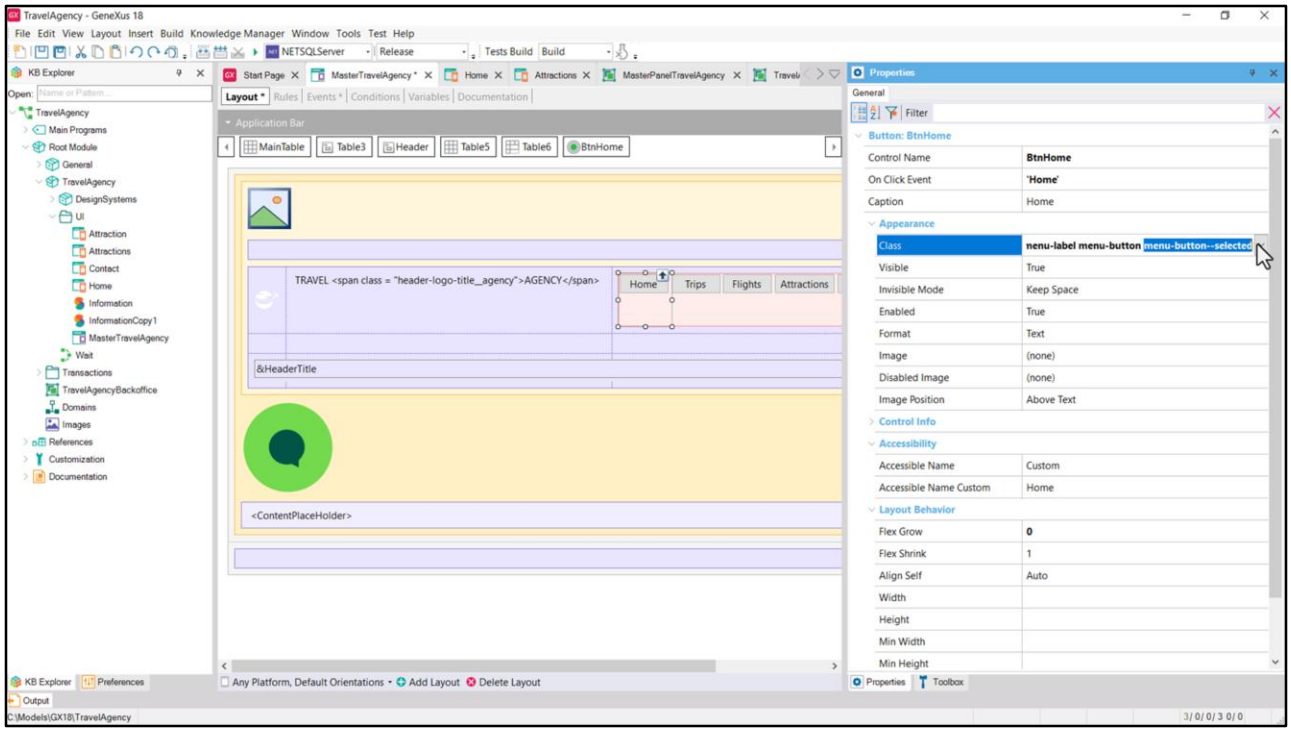


Let's see it in the first case:

- The User Interface of the panel is built based on the properties defined at Design Time (i.e., the static ones),
- And the ClientStart event starts running; again, if it has any asynchronous execution then at that time the panel is rendered, and waits for the asynchronous execution to end. When the asynchronous execution ends, the ClientStart execution continues and at the end, if necessary, it is rendered again.
- If there was no asynchronous execution, the panel is rendered for the first time when the ClientStart event ends.
- Please note: if the ClientStart of the Master Panel did not end when this one did, then here there is a waiting time until it does, and only after that the Start and then Refresh and then Load are executed.

In short, the Start of the Panel doesn't begin until the ClientStart of the Master Panel has ended.

Let's follow this order in detail for our case, paying attention to what happens with the image and the title of the Header and the menu buttons.

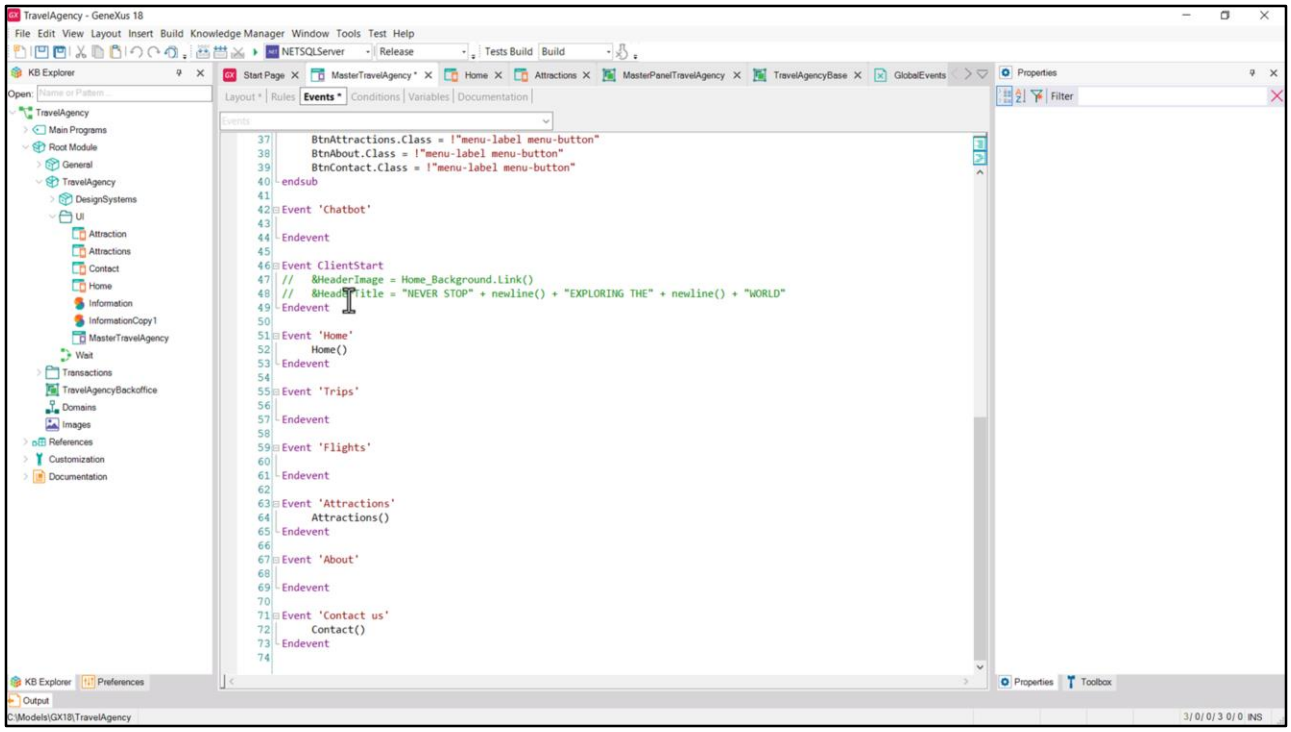


So, we start with the Master Panel, as we were saying, and the static properties are applied first.

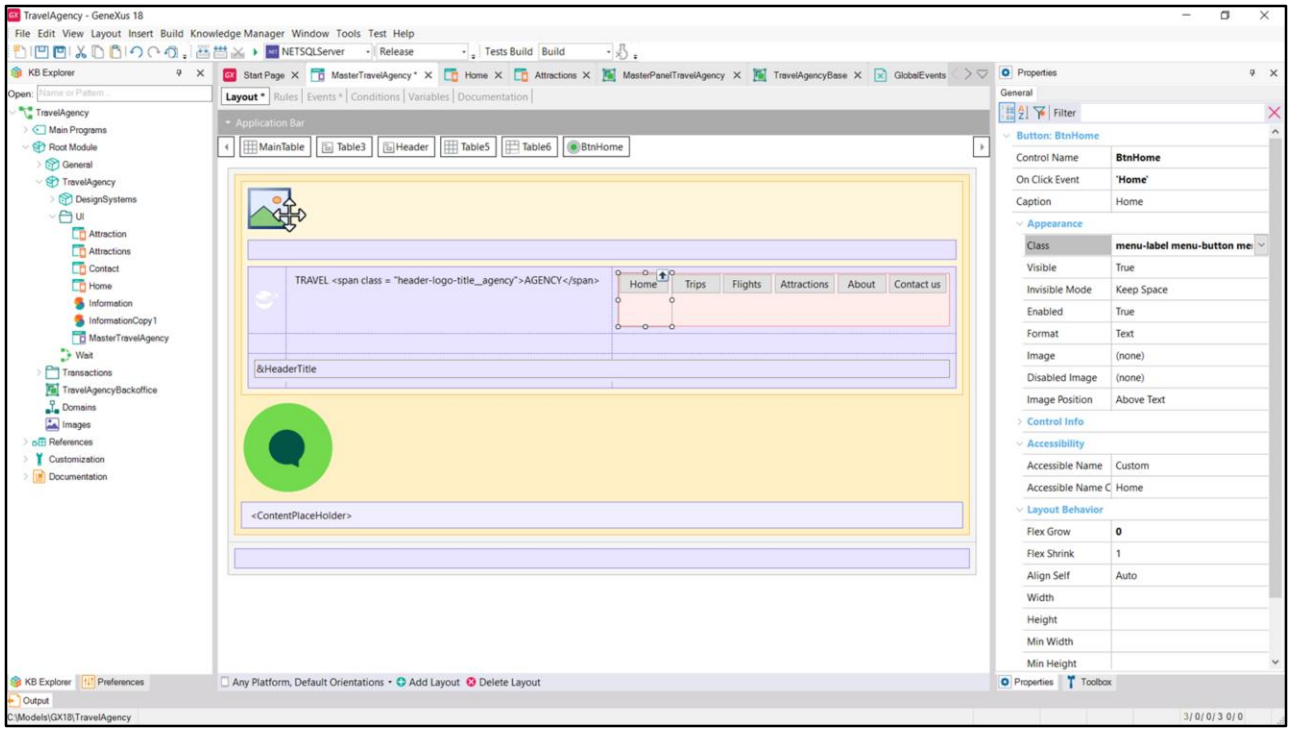
Then the content of the image and title variables will be empty; they will have their properties, the properties, for example, of the classes.

And the menu will also have all these classes associated with each button. In particular, note that for the Home button we also have the selected class, which we had defined in the previous video (we had left it static because we still hadn't started to implement all this dynamic aspect of changing the screen according to the menu). We will have to remove it, clearly. But for now I want to leave it so that we can think about why it should be removed.

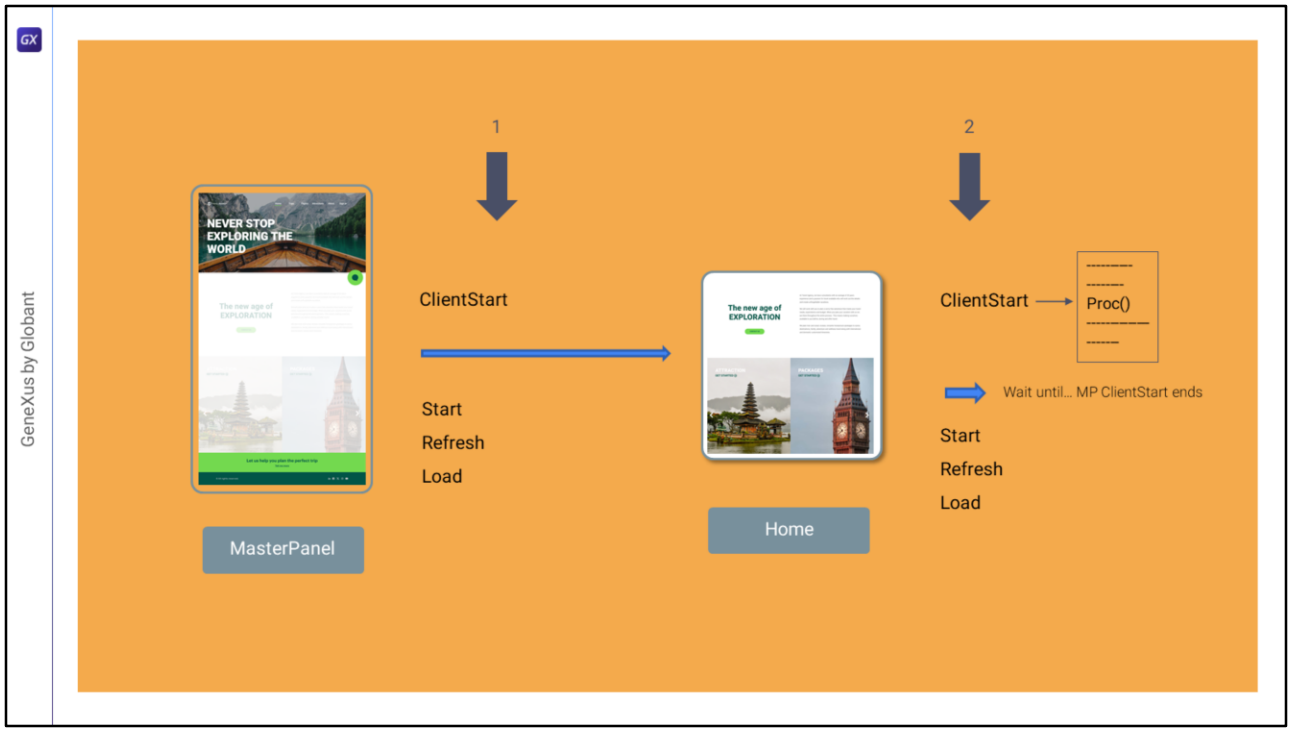
We were saying, then, at this point all the design time properties are taken into account. In particular, the statically defined classes are assigned to our controls.



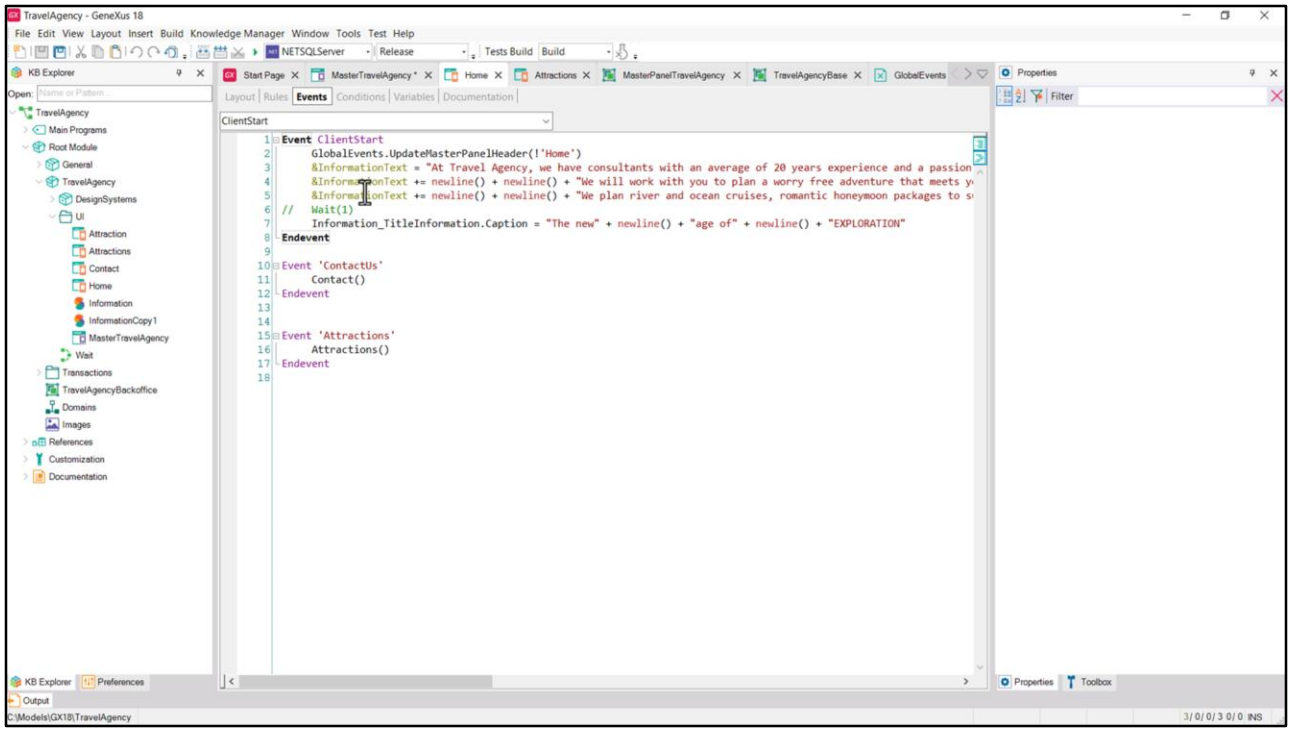
Well, then it will execute, the ClientStart event, which we had commented... And nothing happens there.



As a consequence of all this, the Master Panel is rendered with what we have so far: the image and title variables will be empty, and the menu with the Home option selected.



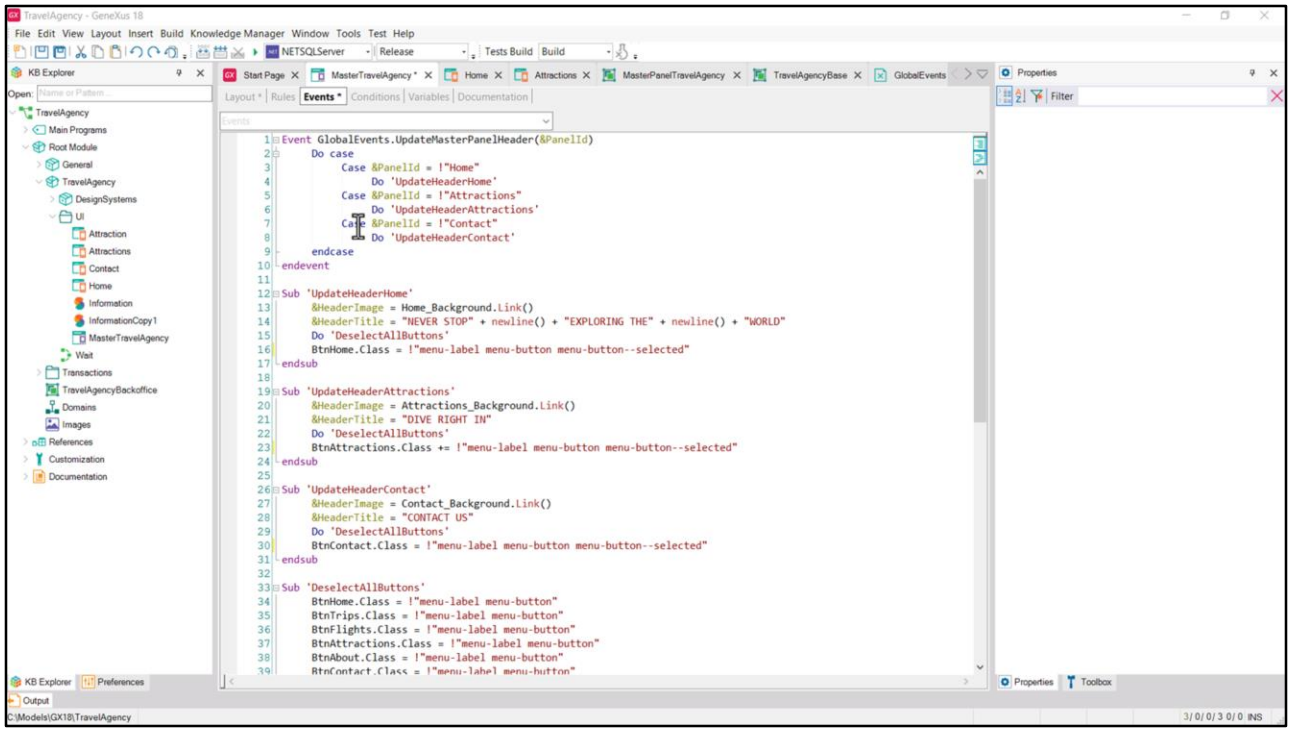
Next, the Panel itself and its ClientStart event are executed.



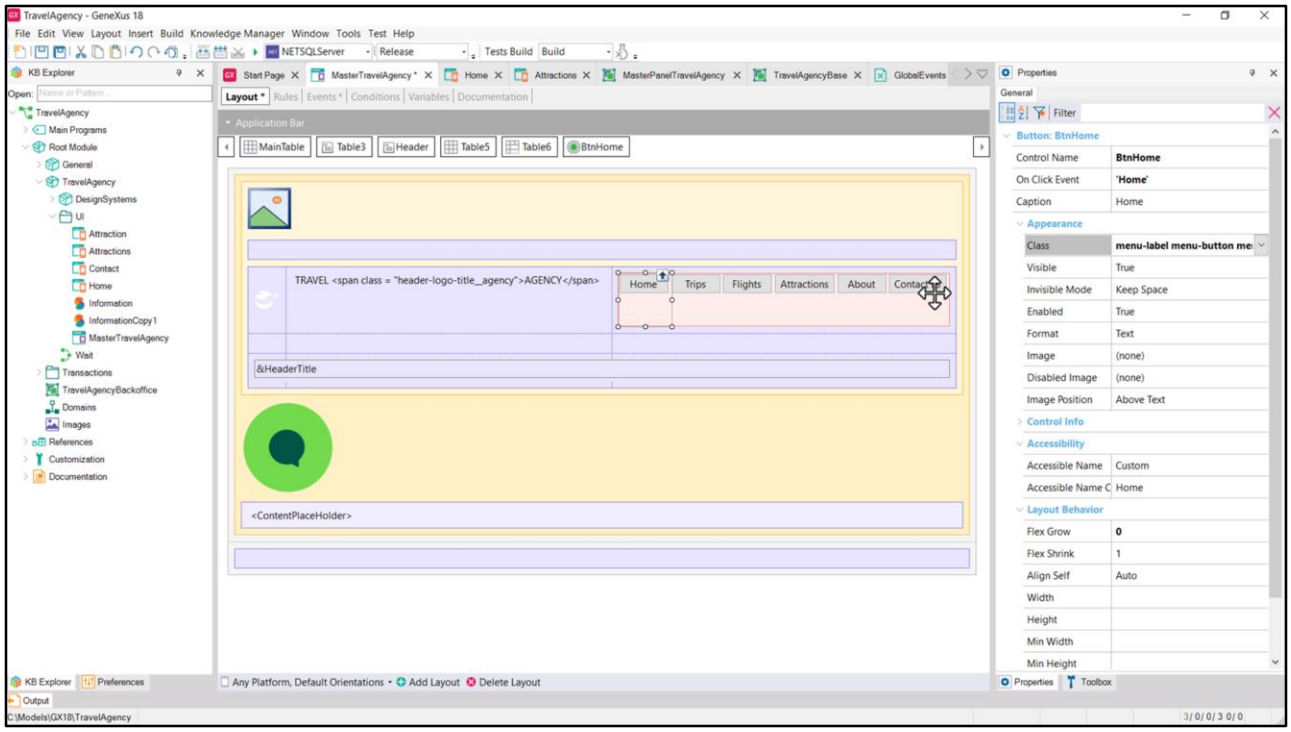
Then the ClientStart event is going to be executed...

If we look at it, it has not only the invocation to the global event, but also these assignments.

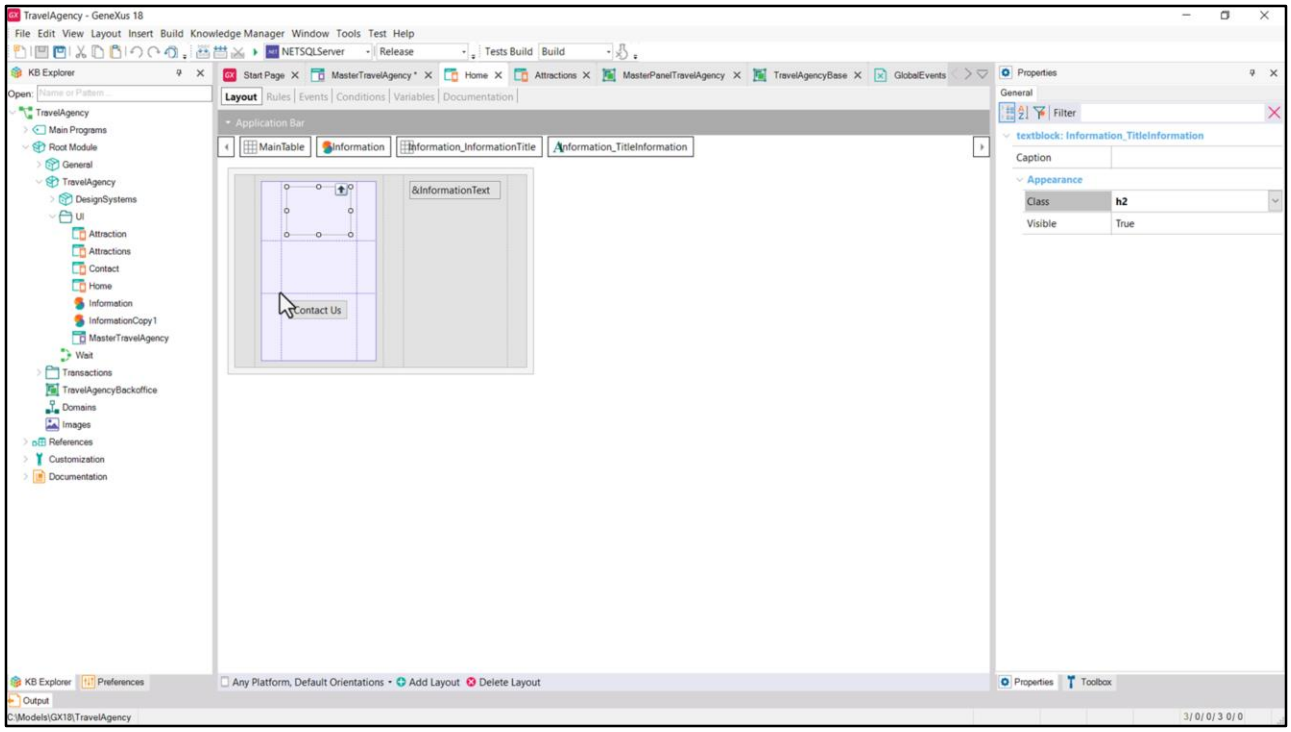
At the time I had my doubts about whether to place the invocation to the global event at the beginning or at the end of the code. The result will be the same, since what will happen is that it will fire the global event asynchronously, but without blocking the execution; that is to say, it will not stop and wait for it to end in order to continue with the next command, but it will continue until it ends.



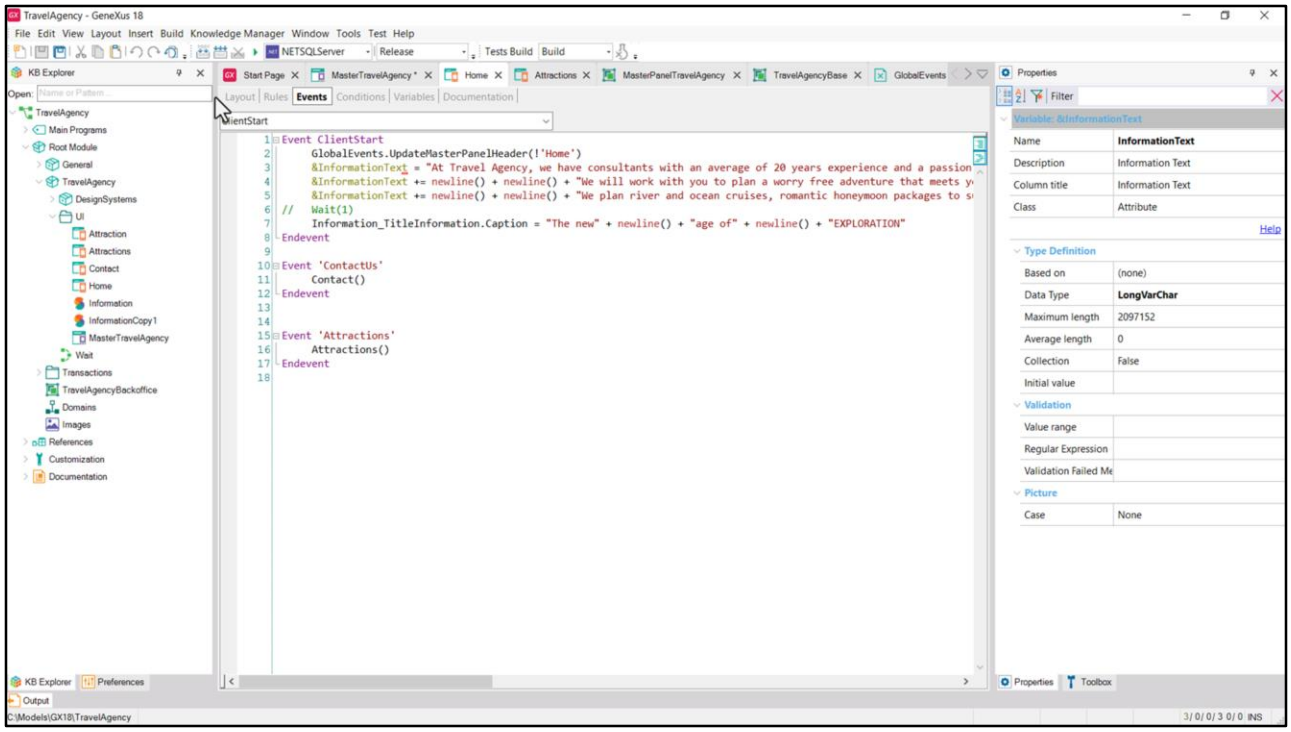
As the global event is being listened to by the Master Panel, which is active on the client in this execution, it will then execute its code.



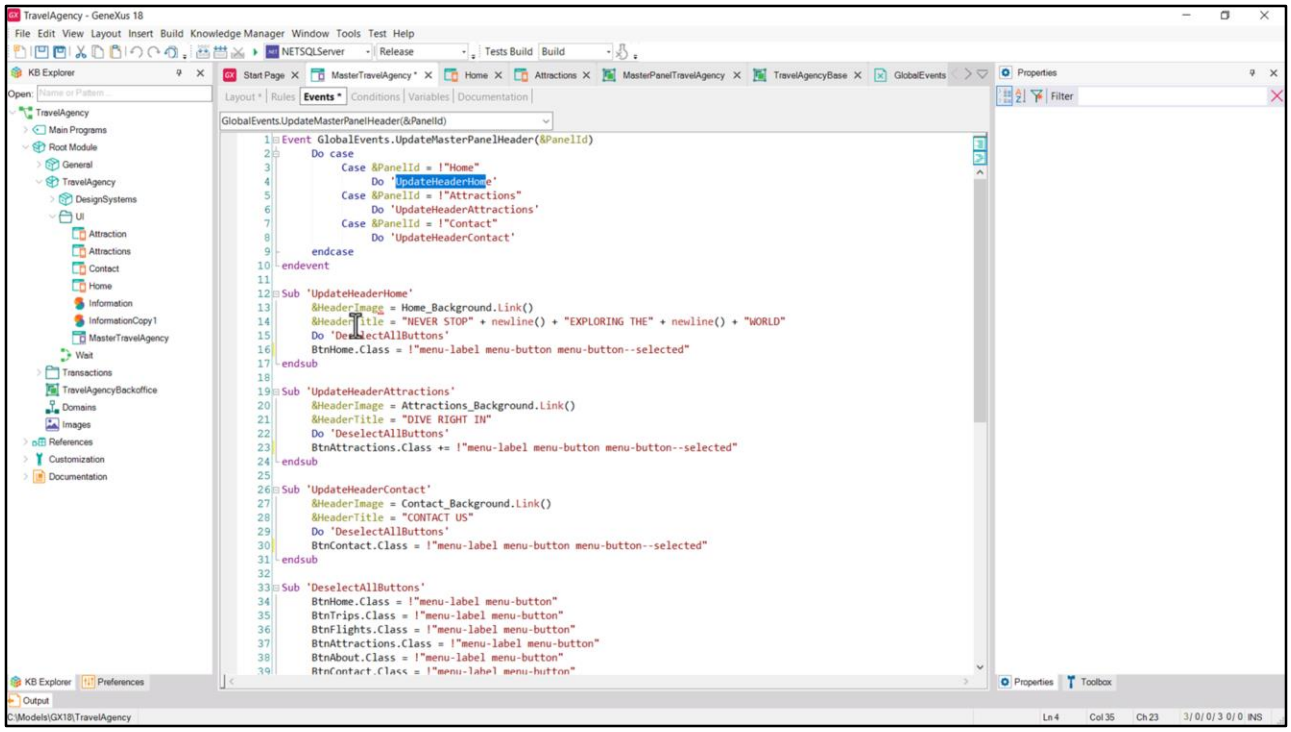
So summarizing the status of the execution up to this point: the Master Panel screen was rendered first, with the empty image and the empty text, but the classes for the menu buttons are applied, so Home would be highlighted in the menu.



These parts of the Home layout were immediately rendered…

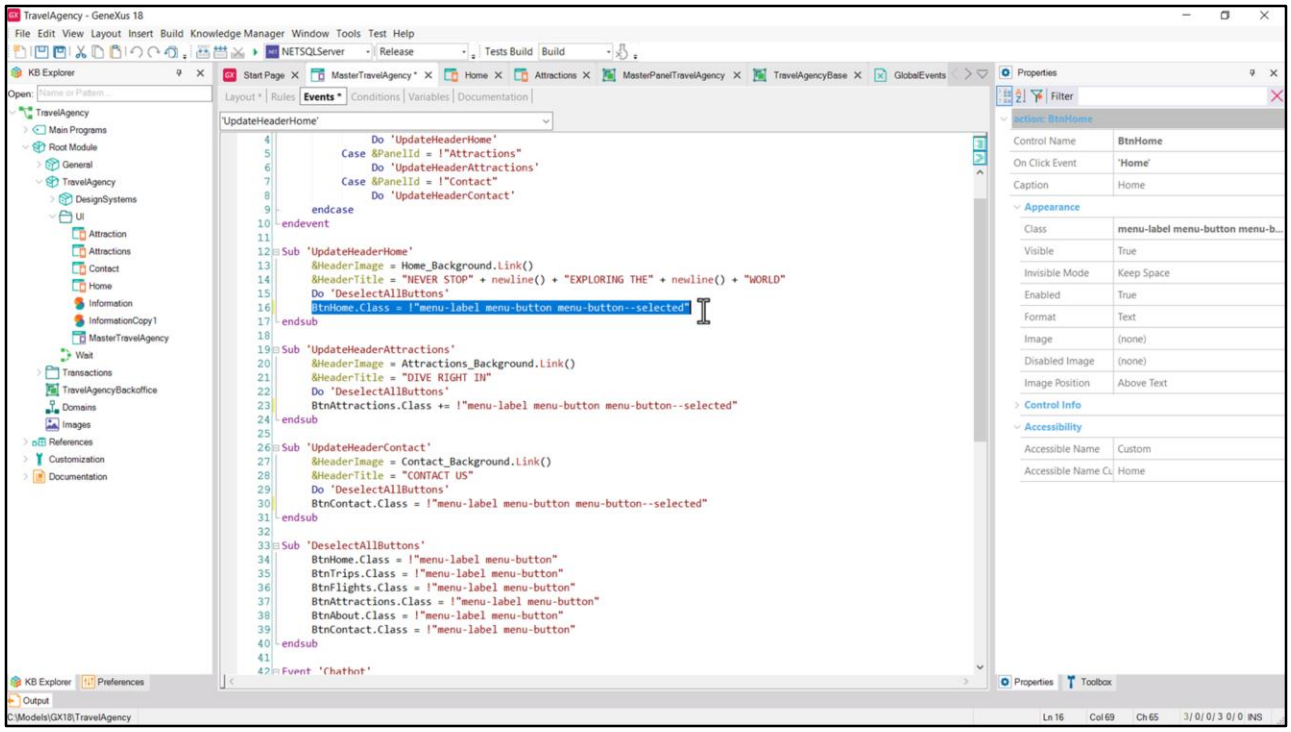


...and in this moment...



...the Do case of the global event is being executed; this event is always a client event.

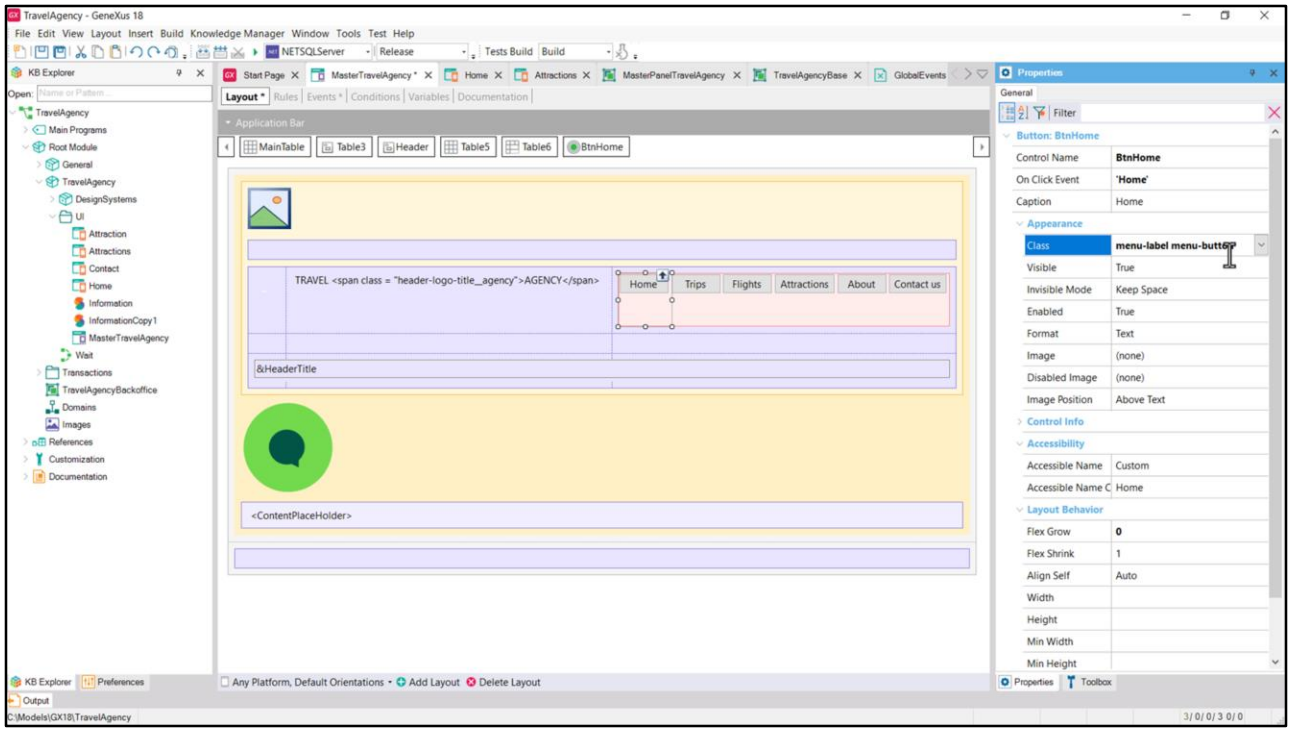
Then this subroutine is executed here... it now assigns a value to the &HeaderImage and &HeaderTitle variables.



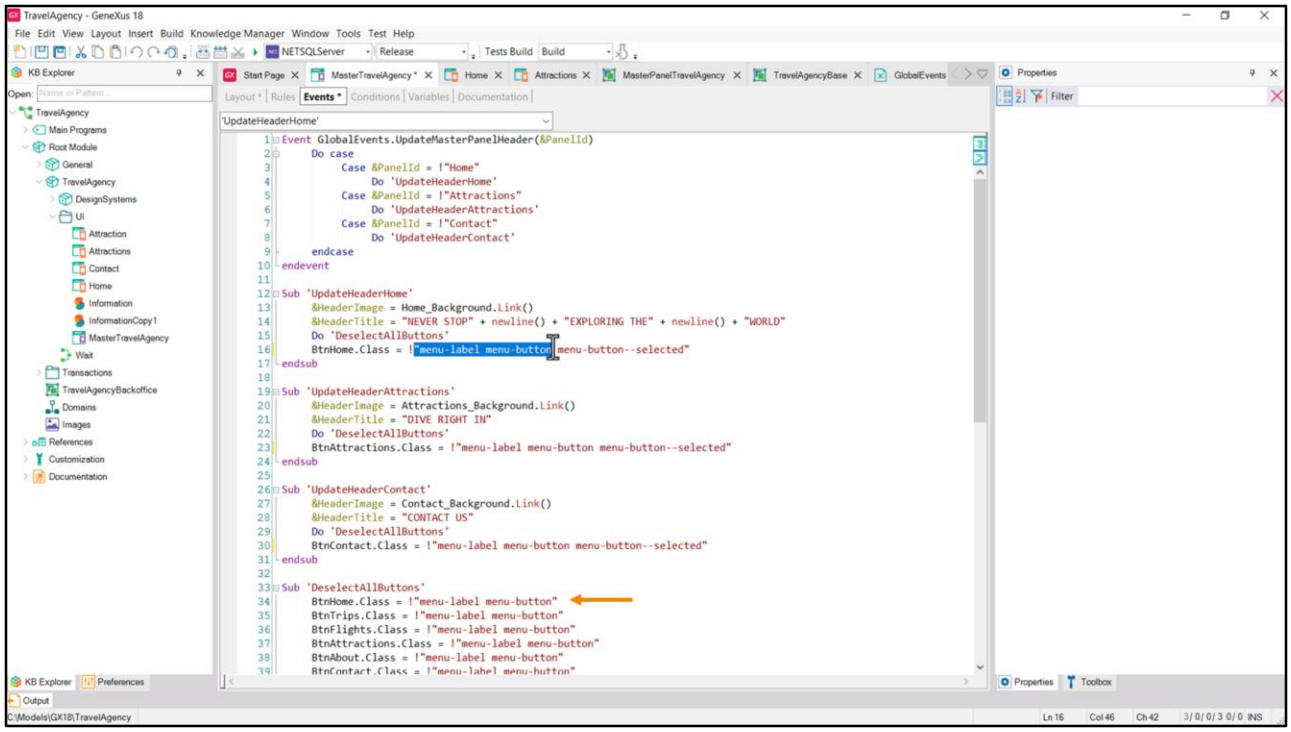
And it overwrites in this other subroutine the classes of all the buttons, so that of BtnHome, which had 3 classes, now has 2. But since we are running the Home panel, it immediately recovers the same 3 classes.
When all this ends, this part of the Header is rendered again.

In this process we made 2 renderings of the Home button. The first one with the static classes, and the second one when triggering the global event. In the case of Home it is perhaps more subtle, but let's think about what would happen if we were loading the Attractions panel, not the Home panel. In the first rendering the Home button will be selected; in the second one, when the global event is executed, the Home button will be deselected and the Attractions button will be selected instead. This could cause a flickering between the first rendering and the second one.

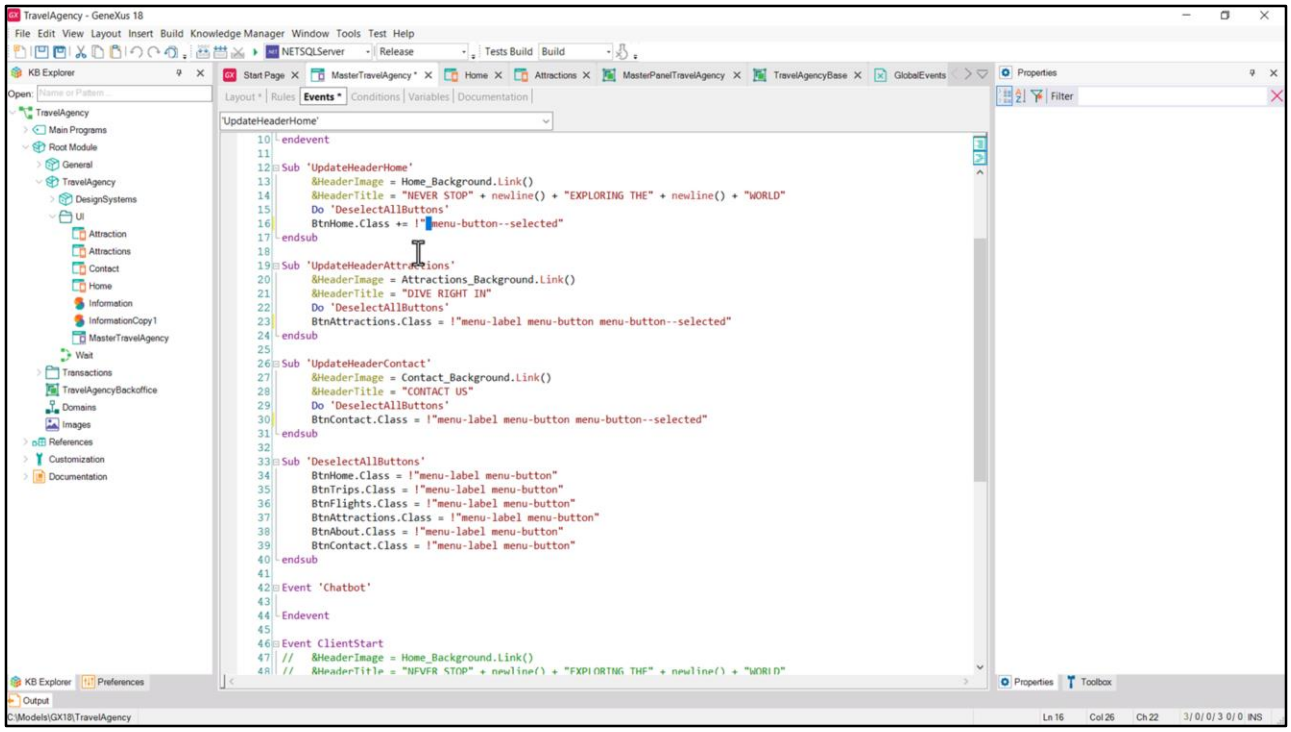
Actually, in this case, it will be rather insignificant because everything is immediately solved in the client.



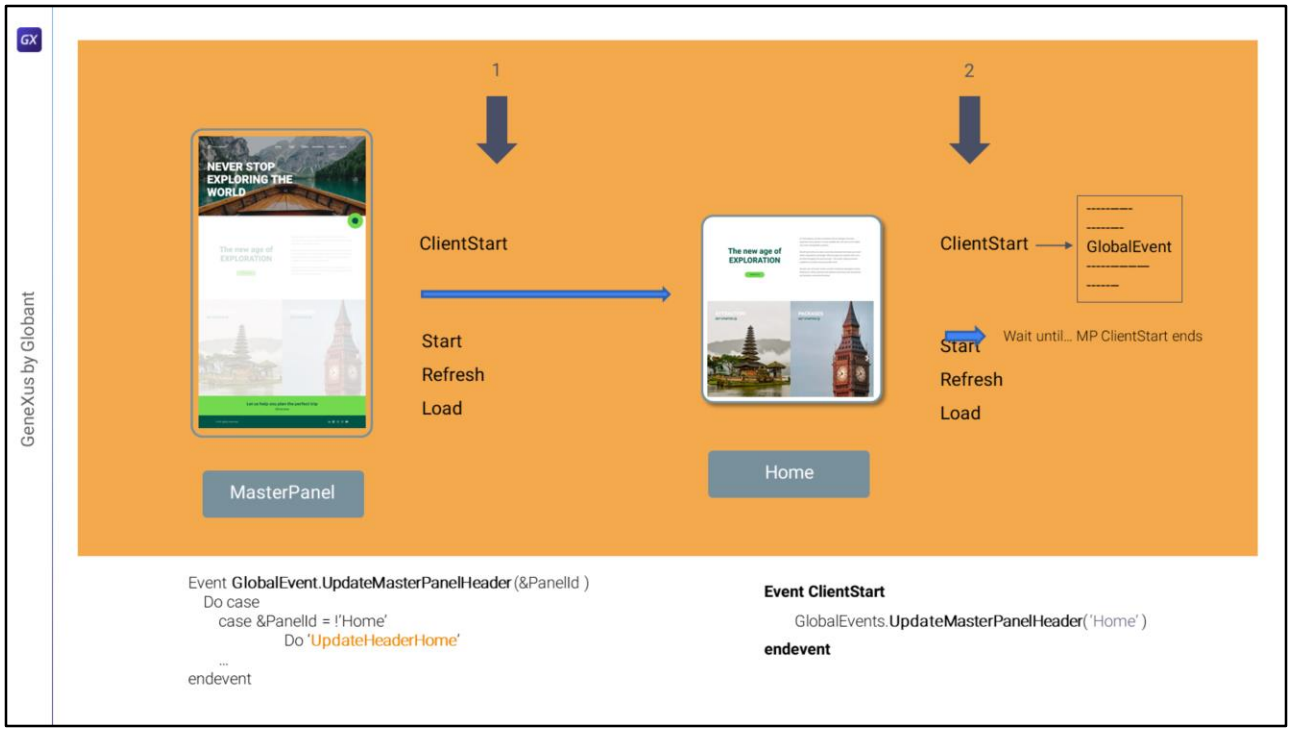
But to make things right, let's remove the static assignment from the selected class of the Home button. In this way, they are all consistent, with the default classes, so that when they are overwritten with the same values in the second rendering, there will be no flickering at all.



Call me obsessive, but this is bothering me a bit... Why am I telling it again that it has these two classes, if it is already specified here?



We're going to make better use of the “plus equals” operator so that, to what it had, it adds what follows. Make sure you leave a blank space here, because this is a string concatenation, and otherwise, this will stick to this. I do the same change for the other two cases and run.



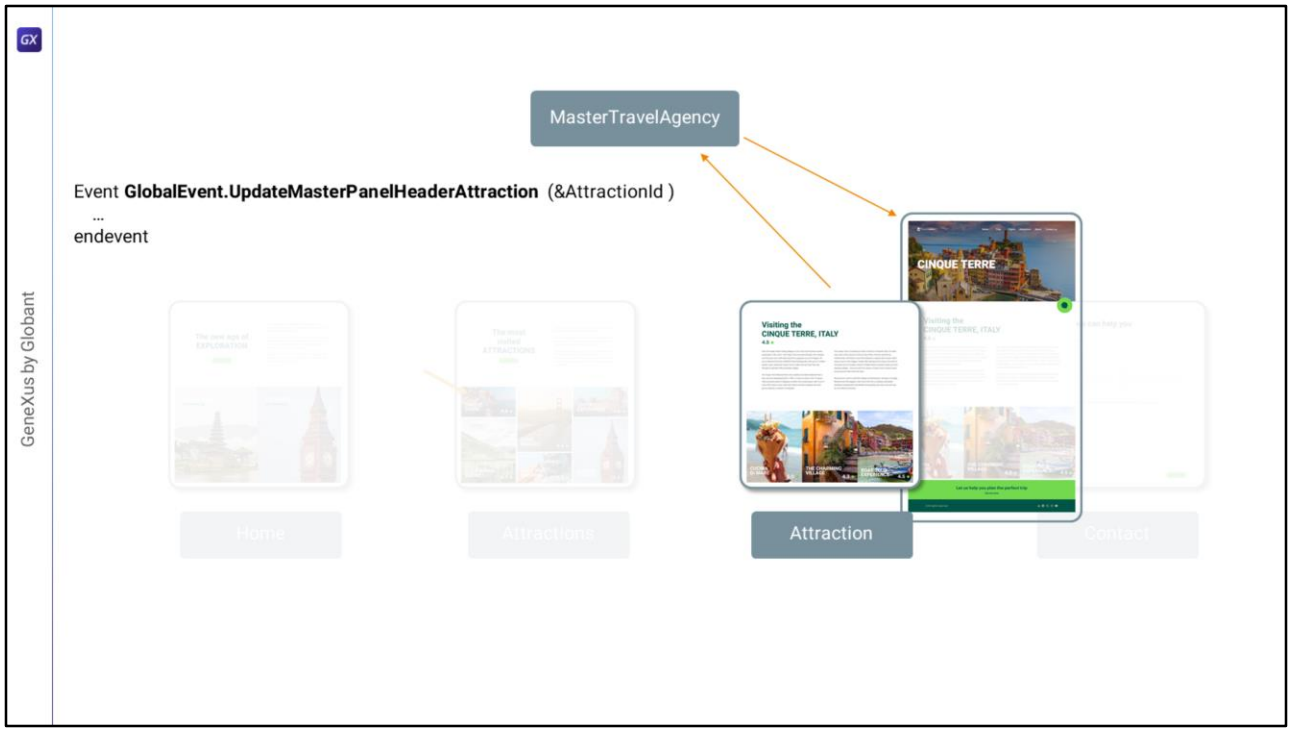
To summarize:

When running our panel, we start with its Master Panel, considering the layout controls according to their static properties. Next, the ClientStart event is executed, which in our case doesn't have any asynchronous command (actually, it doesn't have anything programmed) so the Master Panel is rendered with what we have up to this point.

The order is given to build the panel and project it in the ContentPlaceHolder, which will be done in this concurrent execution line; in parallel, in the first one, the Start, Refresh and Load events are executed, which are empty. So nothing happens.

In the Panel execution line, when it is built, the static properties of its controls are considered, and the ClientStart event is executed. In this case we have the firing of a global event, which is listened to by the Master Panel, so this event is going to be executed concurrently and the execution of the ClientStart event of the Panel will continue, and it will not be stopped by the other one. When it ends, the panel is rendered and since the ClientStart event of the MasterPanel ended some time ago (actually, before launching the build), then it will follow with the execution of Start, Refresh, Load.
In parallel, when the Master Panel executes the global event, as it modifies the User Interface contents, it will render them.

As I already mentioned, we are working so that when navigating between Panels with the same MasterPanel, the MasterPanel is not built again, in order to improve the performance and user experience.



Now we would have to see what is still pending: how to make the Master Panel load in the Header the image and title of the attraction that the Attraction panel received as a parameter when it was executed...

...and that it received it when the user selected a tourist attraction in this grid.

We know it will be from this other global event. So as not to go any further here, we'll do it in the next video. See you in the next video.


GeneXus by Globant
GeneXus™
by Globant
training.genexus.com