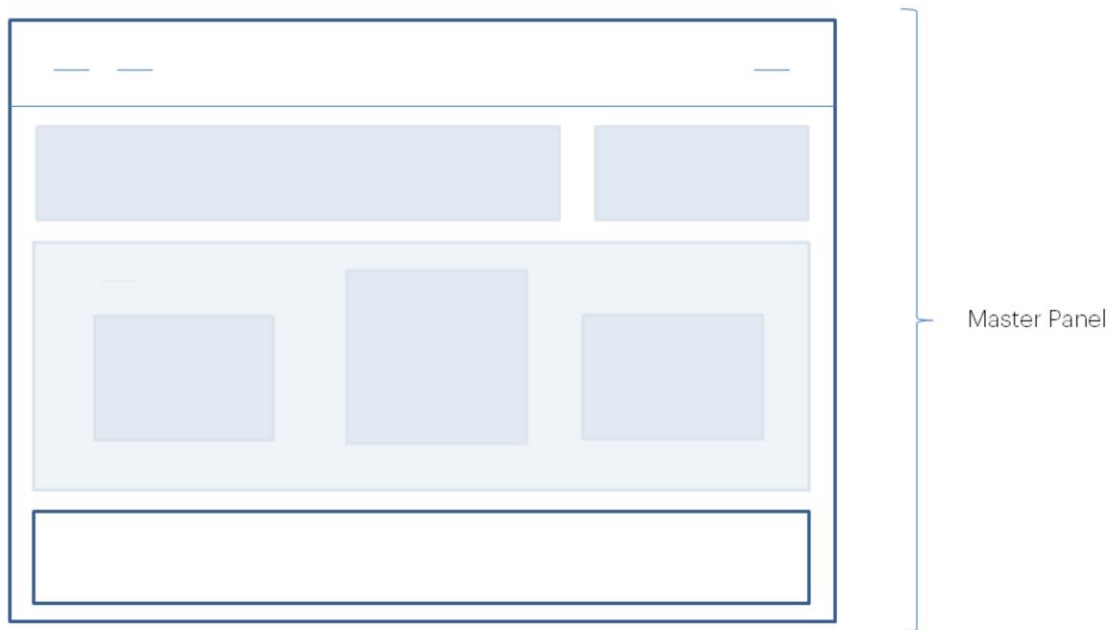


# Global Events

Interaction between components of the same screen

*GeneXus*<sup>™</sup>

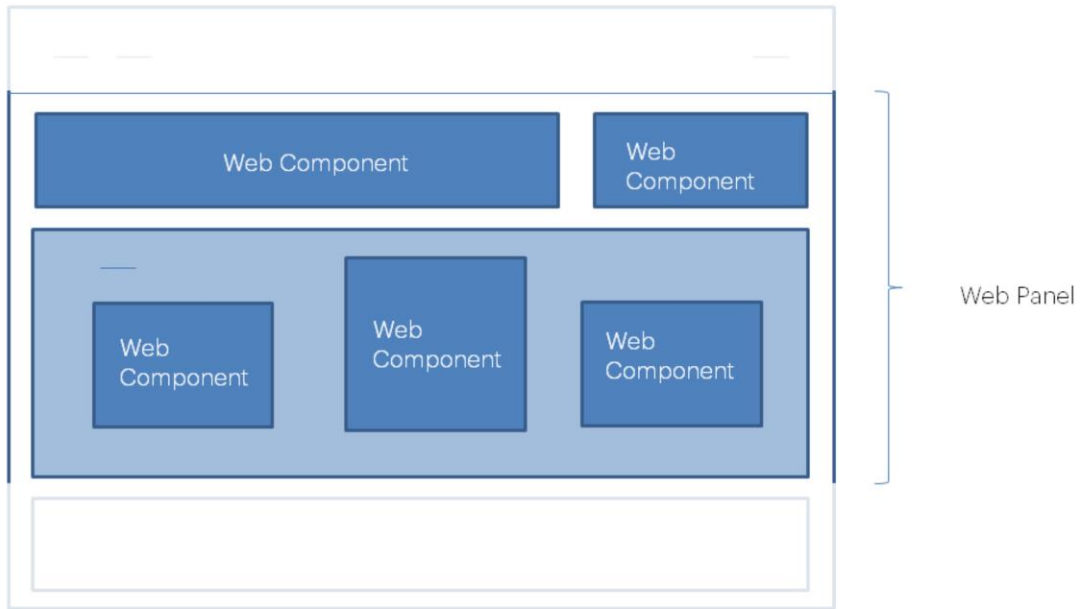
## Screen made up of many screens

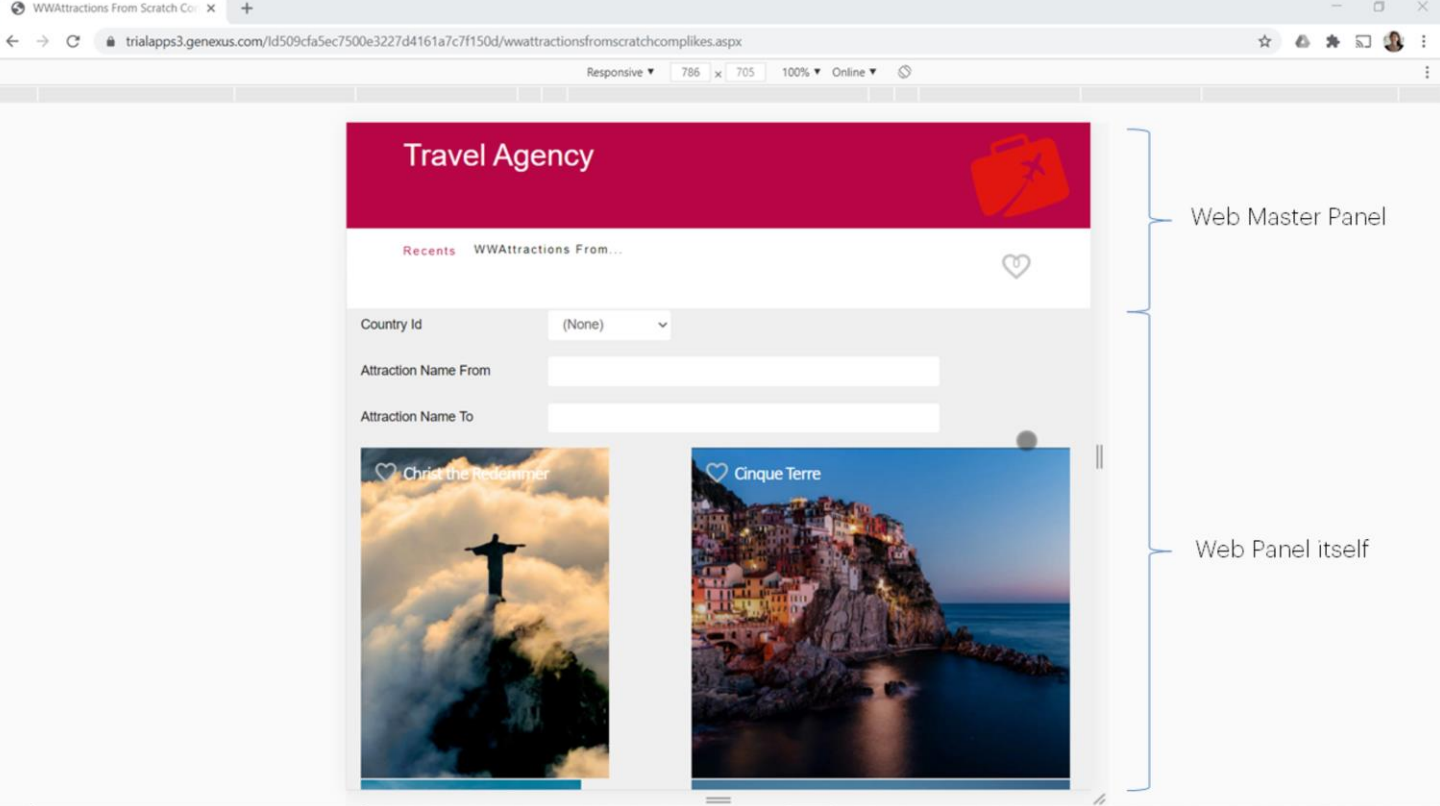


As we know, in order to reuse, we usually componentize as much as possible.

So, if we look at any of our screens in general they correspond to several objects interacting and not just one.

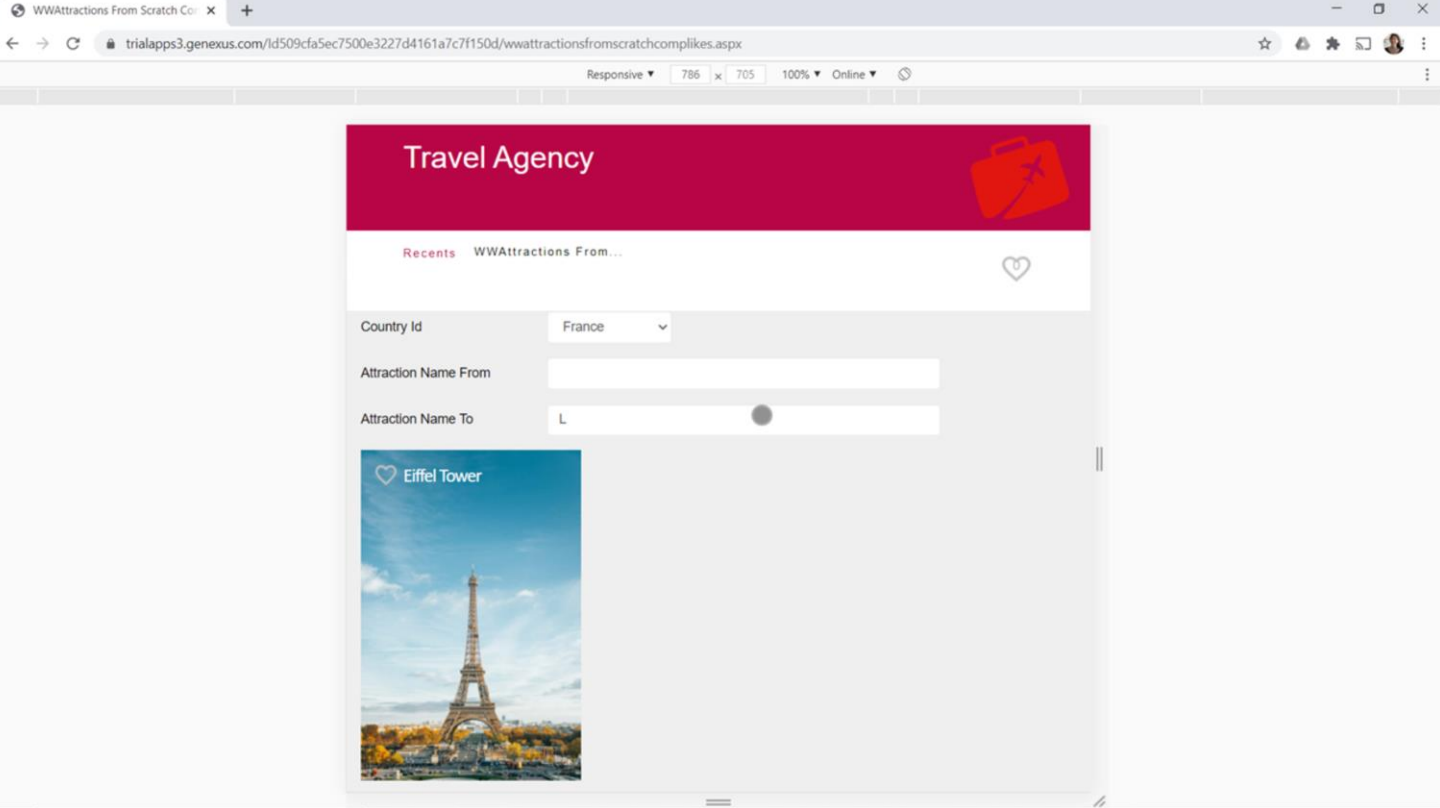
Screen made up of many screens



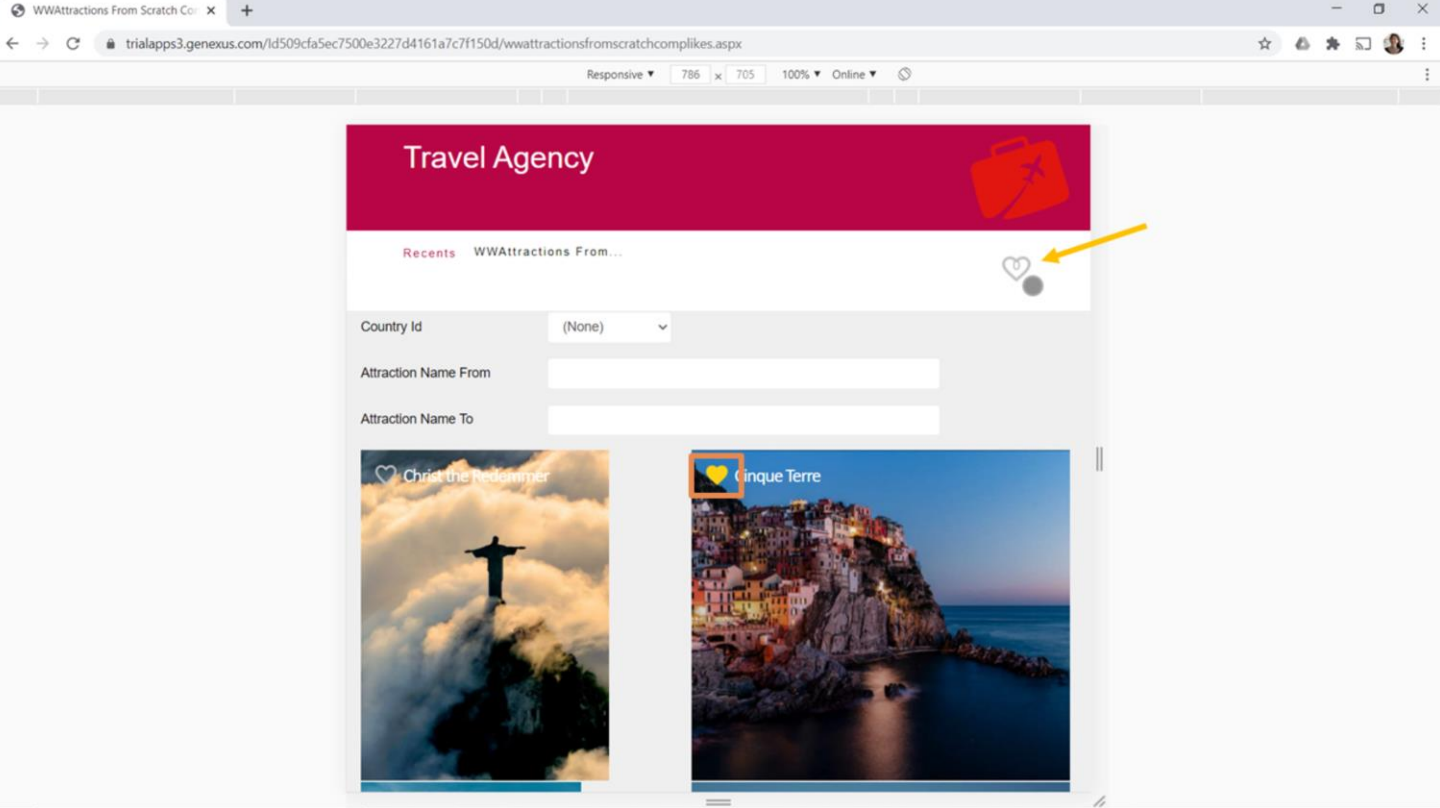


For example, it is the case of a web panel like the one we saw in the initial course of GeneXus, which listed the tourist attractions that a travel agency offered to visit. We will not deal here with the design, which is barely outlined and needs more work.

We are viewing a screen that is not too complex, which will have an entire part coming from the master panel, and its own part. In the latter, some parts are implemented, in turn, as components, although this is not noticeable at runtime.



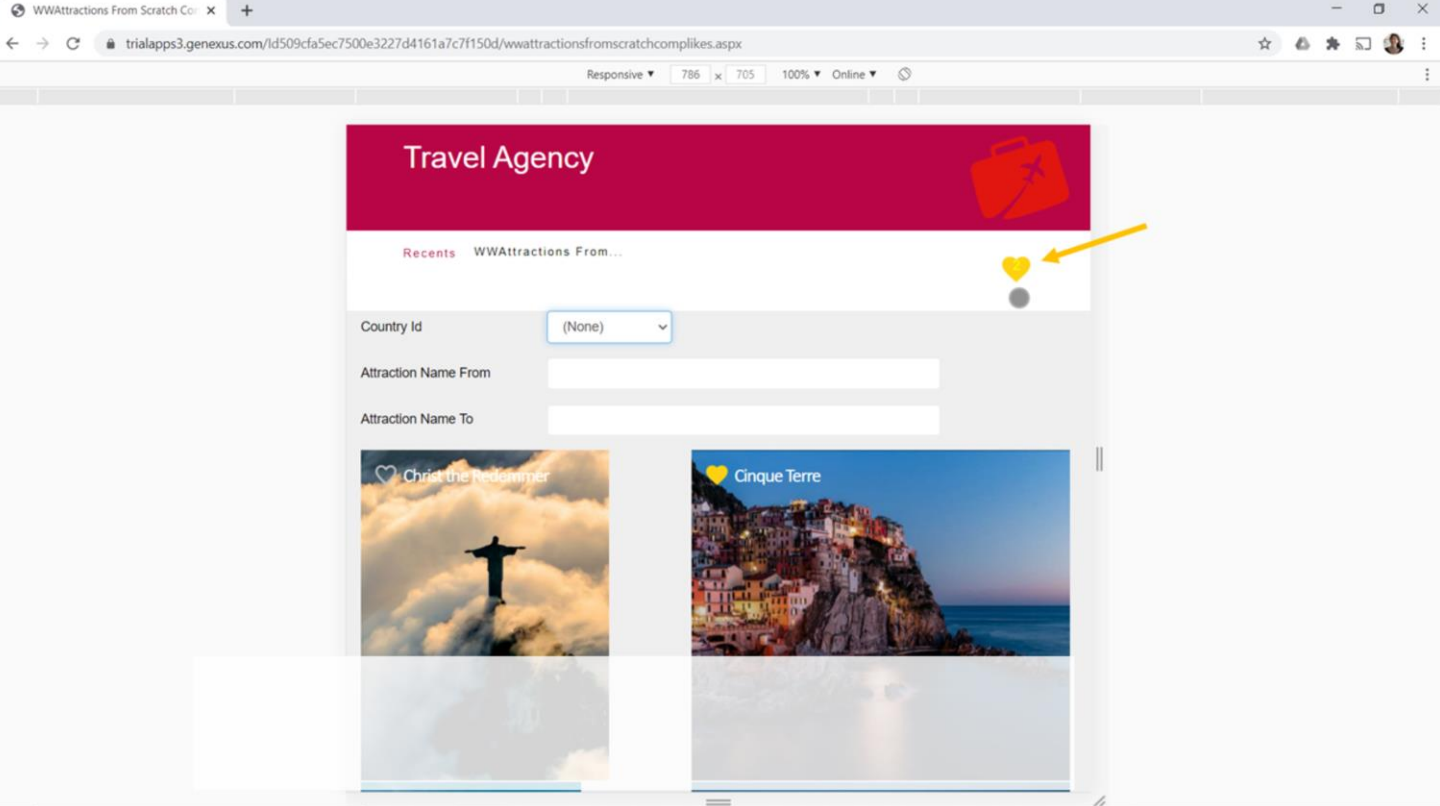
We can filter by country, and also by name of attraction.



But what is most interesting, and what will allow us to introduce the topic of this video, is that we have added the possibility of selecting tourist attractions as favorites.

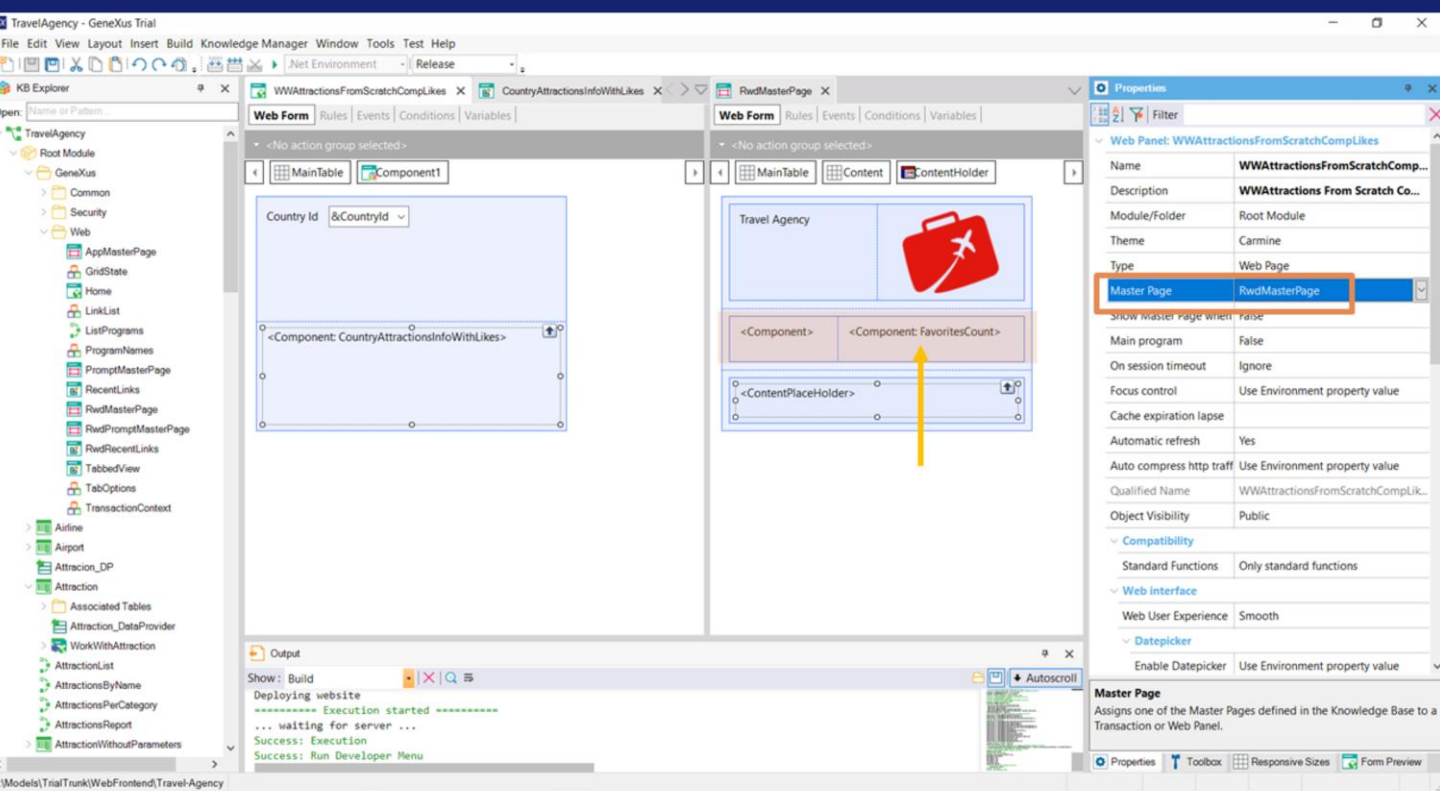
For example, I select this one, and this one. In other words, we have marked two favorite attractions.

However, if we look up here, it shows zero, as if we didn't have any favorites.



We refresh the screen, and now we see this number 2. If I remove a favorite, this number should be refreshed to 1.

Our guiding question is how do we get that value to update automatically as each attraction is selected or deselected as favorite?



We go to GeneXus to see this Web Panel.

We see, first of all, that it has an associated master panel, which is the default. Therefore, the content of its screen will be loaded in the content place holder of the master page.

And out of it will be all this other stuff.

In particular we have two components: a default one, which implements the recent links, and another one that we have added to show the total number of tourist attractions that the user has selected as favorite.



The screenshot displays the GeneXus IDE interface. On the left, the 'FavoriteAttraction' structure is shown with a table:

Name	Type	Description	Formula	Nullable
FavoriteAttraction	FavoriteAttraction	Favorite Attraction		
DeviceId	DeviceId	Device Id		No
AttractionId	Id	Attraction Id		No

Below it, the 'ClientInformation [Read-only]' structure is shown with a table:

Structure	Type	Is Collection	Description
ClientInformation			Client Information
Id	VarChar(128)	<input type="checkbox"/>	
OSName	VarChar(40)	<input type="checkbox"/>	
OSVersion	VarChar(40)	<input type="checkbox"/>	
Language	Character(20)	<input type="checkbox"/>	
DeviceType	SmartDeviceType, G...	<input type="checkbox"/>	
PlatformName	VarChar(128)	<input type="checkbox"/>	
AppVersionCode	VarChar(40)	<input type="checkbox"/>	
AppVersionName	VarChar(40)	<input type="checkbox"/>	
ApplicationId	VarChar(128)	<input type="checkbox"/>	

On the right, the 'Properties' window for the 'Attribute: Deviceld' is shown:

Name	Deviceld
Description	Device Id
Title	Device Id
Column title	Device Id
Contextual Title	Id
Formula	
Nulls in Forms	Empty as Null
Class	Attribute
Qualified Name	Deviceld
<b>Type Definition</b>	
Supertype	
Based on	Deviceld:Domain
Data Type	VarChar
Maximum length	128
Average length	0

To save the attractions that a user has selected as favorite, 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) then we can temporarily keep favorites per browser instance.

For this, 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...

ClientInformation 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. In the table of this transaction we will save the favorites.

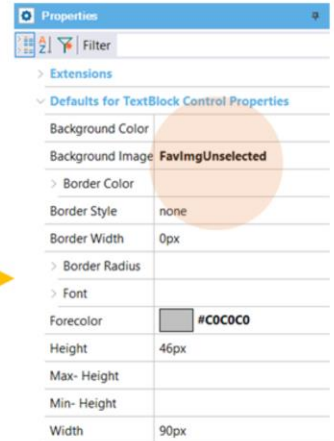
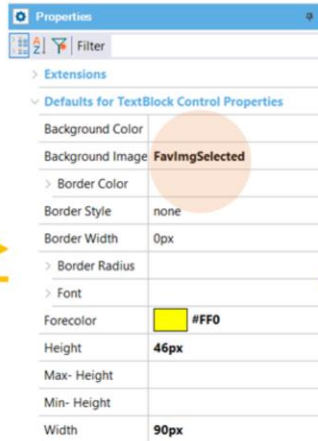
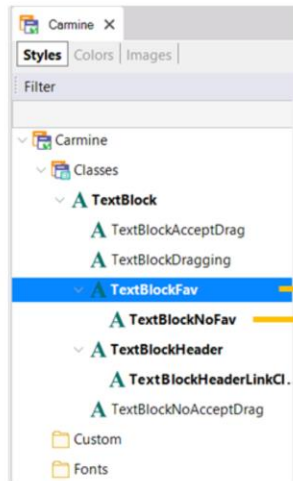
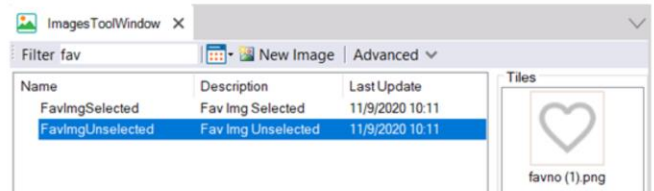
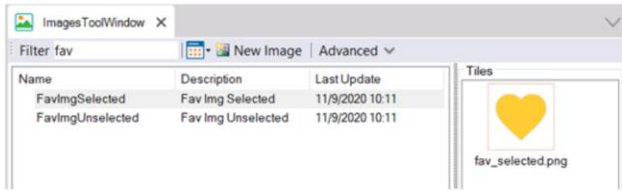
The screenshot displays the GeneXus IDE interface. On the left, a 'Web Form' is shown with a 'MainTable' containing a 'Travel Agency' component, a 'FavoritesCount' component, and a 'ContentPlaceHolder'. A yellow arrow points from the 'FavoritesCount' component to the right pane. The right pane shows a 'Web Form' with a 'MainTable' and a 'TxtCount' component. Below the 'TxtCount' component, the code for the 'Event Start' and 'Sub 'GetCount'' is visible. The 'Event Start' code is 'Do 'GetCount'' and 'Endevent'. The 'Sub 'GetCount'' code is: '&Amount = Count(AttractionId, DeviceId = ClientInformation.Id, 0); txtCount.Caption = &Amount.ToString().Trim()'. Below this, there is an 'If &Amount.IsEmpty()' block with 'txtCount.Class = ThemeClass:TextBlockNoFav' and an 'else' block with 'txtCount.Class = ThemeClass:TextBlockFav'. The code ends with 'endif' and 'endSub'. The right pane also shows the 'Properties' window for the 'textblock: TxtCount' component, with the 'Caption' property set to 'Text Block'.

If we look at the Web component being loaded here, in the web Master Panel, we see that it only contains a text block with this name, whose caption is loaded at runtime, initially 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.

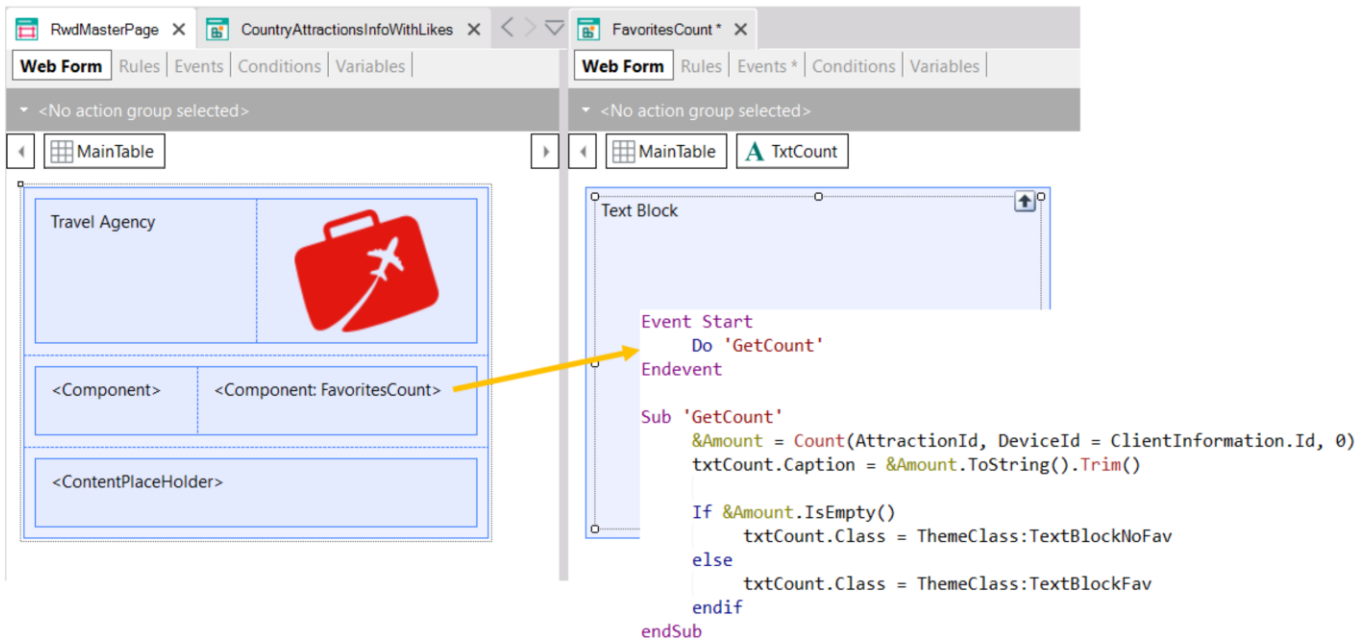
And to show it with the favorite image and in the right color and format, we have added two classes to the corresponding theme –which in this case is Carmine.

If the number of attractions is zero, to the text block we assign the class that designs the empty favorite and otherwise, the one that designs the filled yellow favorite.



For this we had to previously insert two images in the KB...

If we look at the classes in the theme, the Background Image property shows the filled yellow heart and for this other class the empty gray heart.



```
Event Start
  Do 'GetCount'
Endevent

Sub 'GetCount'
  &Amount = Count(AttractionId, DeviceId = ClientInformation.Id, 0)
  txtCount.Caption = &Amount.ToString().Trim()

  If &Amount.IsEmpty()
    txtCount.Class = ThemeClass:TextBlockNoFav
  else
    txtCount.Class = ThemeClass:TextBlockFav
  endif
endSub
```

So this text block calculates the number and displays it with the appropriate background image and colors.

This Start event will be triggered when the master panel is opened.

The screenshot displays the GeneXus IDE interface with two web forms open. The left form, 'CountryAttractionsInfoWithLikes', contains a 'Travel Agency' section with a red suitcase icon, a '<Component>' placeholder, and a '<Component: FavoritesCount>' component. The right form, 'WWAttractionsFromScratchComplikes', features a dropdown menu for '&CountryId' and a '<Component: CountryAttractionsInfoWithLikes>' component. A yellow arrow points from a '<ContentPlaceholder>' in the left form to the component in the right form. The Properties window on the right shows details for 'Web Component: Component1', including 'Control Name: Component1', 'Object: CountryAttractionsInfoWithLikes', and 'Parameters: &CountryId'.

```

Event &CountryId.ControlValueChanged
    Component1.Object = CountryAttractionsInfoWithLikes.Create(&CountryId)
Endevent

```

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.

The screenshot displays the GeneXus IDE interface for a web component named "CountryAttractionsInfoWithLikes". The main design area shows a form with two input fields: "Attraction Name From" (containing "&AttractionNameFrom") and "Attraction Name To" (containing "&AttractionNameTo"). Below these is a "GRID" component, which is highlighted with a yellow border. The grid contains a landscape image, a heart icon, and a text field labeled "AttractionName". At the bottom of the form is an "Attraction Id" field with the value "AttractionId".

The "Properties" window on the right shows the following details for the "Web Component: CountryAttractionsInfoWithLikes":

Name	CountryAttractionsInfoWithLikes
Description	Country Attractions Info With Lik...
Module/Folder	Root Module
Theme	Carmine
Type	Component
URL access	No
Show Master Page wh	
Main program	
On session timeout	
Focus control	
Cache expiration laps	
Automatic refresh	
Auto compress http tr	
Qualified Name	
Object Visibility	

The "Grid2's Conditions" dialog box is open, showing the following conditions:



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

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 filters on the screen itself.

Attraction Name From

Attraction Name To

GRID

Attraction Id

**Properties**

General Class

Filter

Free Style Grid: Grid2

Control Name	Grid2
Collection	
Save State	False
Base Trn	Attraction
Order	CountryId, AttractionName whe...
Conditions	CountryId = &CountryId when ...
Unique	
Allow Drop	False
Allow Drag	False
Notify Context Change	False
Tooltip Text	
Width	
Height	
Cell Padding	1
Cell Spacing	2

Event Grid2.Load

```

For each FavoriteAttraction
  where DeviceId = ClientInformation.Id
  ImgFavToggle.FromImage(FavImgSelected)
when none
  ImgFavToggle.FromImage(FavImgUnselected)
endfor
Endevent

```

> Control Info

Class	FreeStyleGrid
Rows	<unlimited>
Custom Render	FlexGrid
Empty Grid Text	

**Properties**

General Class

Image: ImgFavToggle

Control Name	ImgFavToggle
Image	FavImgUnselected
Low Resolution Image	(none)
Return On Click	False
On Click Event	

Appearance

Class	FavImage
-------	----------


We have chosen a Flex grid, which contains a Canvas table, in order to overlap the tourist attraction photo with its name and an image to allow the user to select or deselect the attraction as 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, for each tourist attraction to be loaded this For Each command is executed, which searches in the subordinate table, FavoriteAttraction, if there is a record for this device, the one that is running. If it exists, it means that the attraction is a favorite, and therefore the image with a filled heart is loaded; otherwise, the empty heart is loaded.

Attraction Name From

Attraction Name To

GRID





Attraction Id

```

Event ImgFavToggle.Click
  &isFavorite = ToggleFavorite(AttractionId)
  If &isFavorite
    ImgFavToggle.FromImage(FavImgSelected)
  else
    ImgFavToggle.FromImage(FavImgUnselected)
  endif
Endevent

```

ToggleFavorite X

Source | Layout | Rules | Conditions | Variables

Subroutines

```

1 For each FavoriteAttraction
2   where DeviceId = ClientInformation.Id
3   where AttractionId = &AttractionId
4   &isFavorite = False
5   Delete
6 when none
7   &isFavorite = True
8 new
9   DeviceId = ClientInformation.Id
10  AttractionId = &AttractionId
11 endnew
12 endfor

```

But from all this, what really matters to us is something else: programming of the Click event on this image.

Once the attractions grid is loaded, when the user clicks on an attraction favorite, we call a procedure that checks if the attraction was already a favorite, in which case it removes it from the table and indicates the operation to switch the image for the empty one. And if it's the other way around, it does the opposite, which is to insert.

By doing this, the favorite total should be updated, but it is not.



CountryAttractionsInfoWithLikes \* X

Web Form Rules Events Conditions Variables

<No action group selected>

Grid2Table Table2 Table3 ImgFavToggle

Attraction Name From &AttractionNameFrom

Attraction Name To &AttractionNameTo

GRID

AttractionName

Attraction Id &AttractionId

```

Event ImgFavToggle.Click
  &isFavorite = ToggleFavorite(AttractionId)
  If &isFavorite
    ImgFavToggle.FromImage(FavImgSelected)
  else
    ImgFavToggle.FromImage(FavImgUnselected)
  endif
Endevent

```

WWAttractionsFromScratchCompLikes X

Web Form Rules Events Conditions Variables

<No action group selected>

MainTable Component1

Country Id &CountryId

<Component: CountryAttractionsInfoWithLikes>

FavoritesCount \* X

Web Form Rules Events Conditions Variables

<No action group selected>

MainTable TxtCount

Text Block

RwsMasterPage X

Web Form Rules Events Conditions Variables

<No action group selected>

MainTable Content ContentHolder

Travel Agency

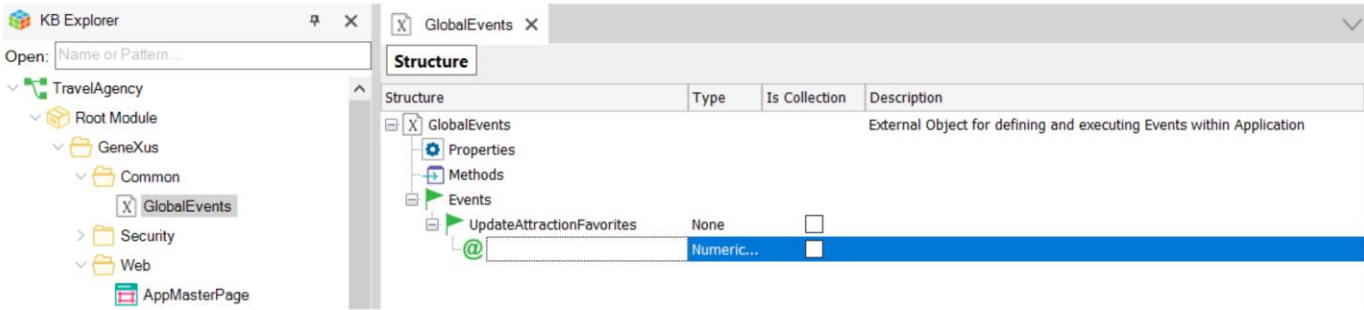
<Component> <Component: FavoritesCount>

<ContentPlaceholder>

In short... this event is triggered in a web component inside this web panel, which in turn runs inside this Web Master Panel, where the component we want to be refreshed is also created.

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.

## Global Events: default external object



To enable the interaction, all KBs will come with this predefined object, **Global Events**, so we can modify it.

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

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

Attraction Name From

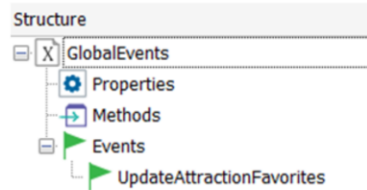
Attraction Name To

GRID





Attraction Id



Event `ImgFavToggle.Click`

```

&isFavorite = ToggleFavorite(AttractionId)
If &isFavorite
    ImgFavToggle.FromImage(FavImgSelected)
else
    ImgFavToggle.FromImage(FavImgUnselected)
endif
GlobalEvents.UpdateAttractionFavorites()

```

Endevent

With this global form of communication, let's put it to work.

Let's go to the panel that should trigger the event. It will happen every time the image 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.

The screenshot displays the GeneXus IDE interface for a web form named 'FavoritesCount'. The form contains a 'MainTable' and a 'Text Block'. The code editor shows the following logic:

```

Event Start
  Do 'GetCount'
Endevent

Sub 'GetCount'
  &Amount = Count(AttractionId, DeviceId = ClientInformation.Id, 0)
  txtCount.Caption = &Amount.ToString().Trim()

  If &Amount.IsEmpty()
    txtCount.Class = ThemeClass:TextBlockNoFav
  else
    txtCount.Class = ThemeClass:TextBlockFav
  endif
endSub

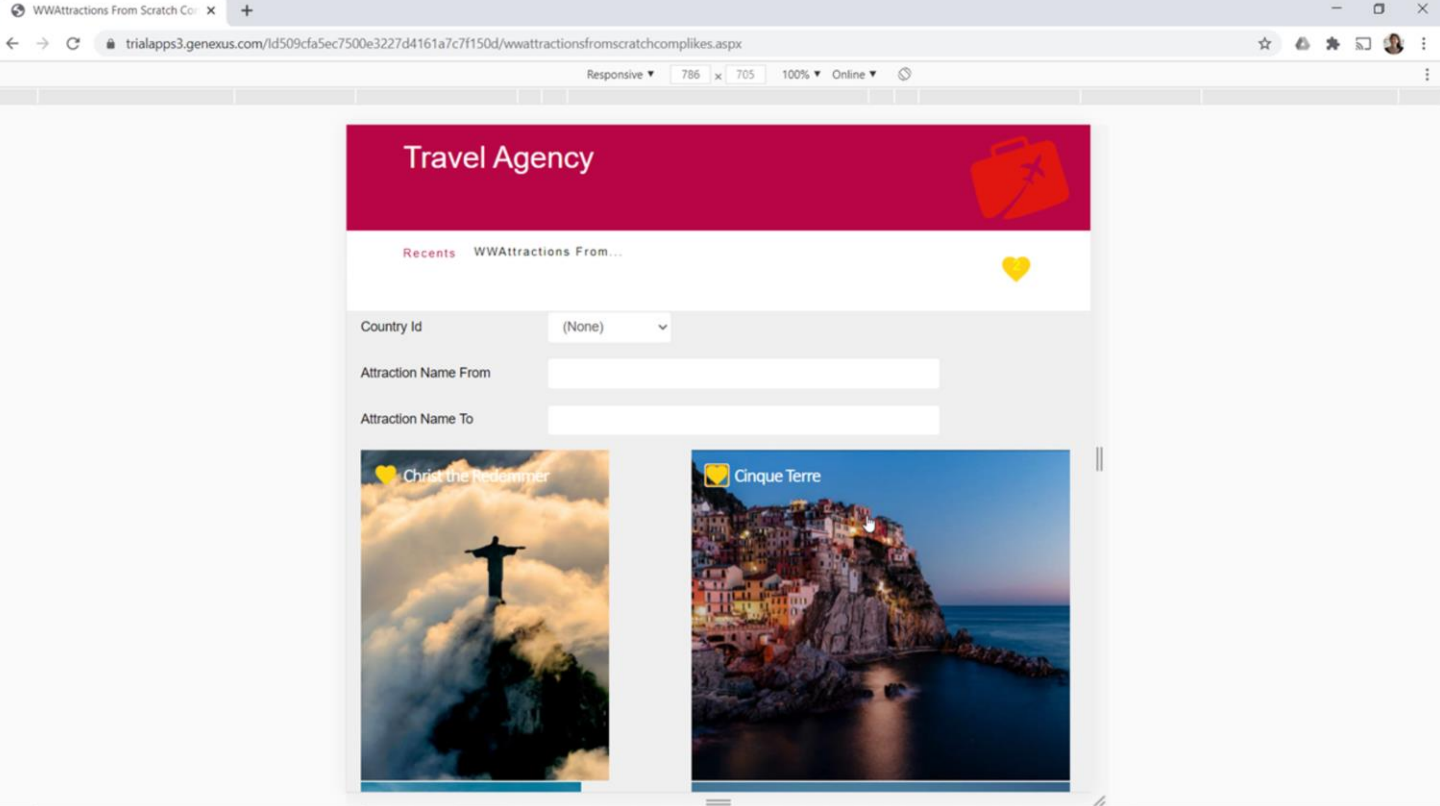
Event GlobalEvents.UpdateAttractionFavorites()
  Do 'GetCount'
endevent

```

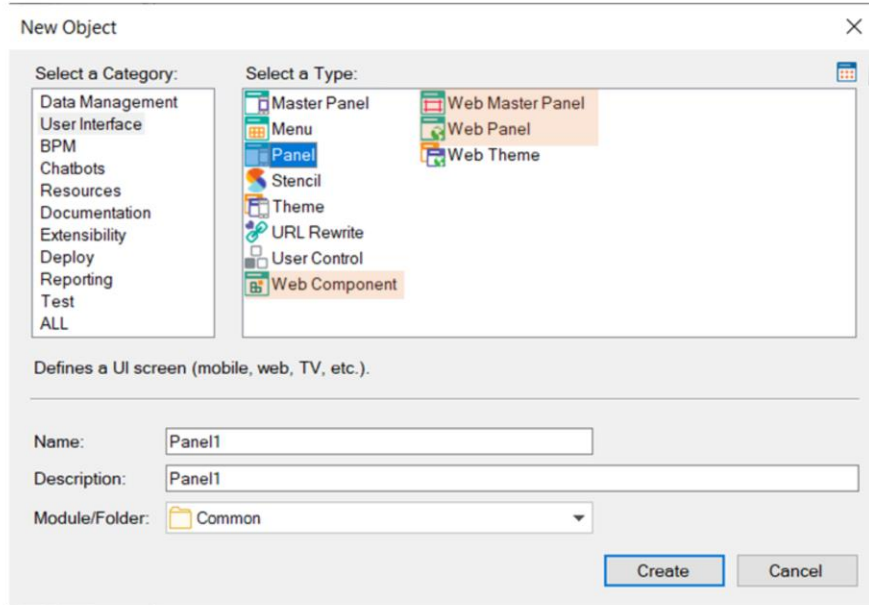
The 'Structure' pane on the right shows the project hierarchy: GlobalEvents, Properties, Methods, and Events, with 'UpdateAttractionFavorites' listed under the Events folder.

Next, what we have to do is to have the component counting the attractions to show them subscribe to this event. In this way, we can listen to it every time it occurs on the screen of the listener.

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



Let's try it. Let's deselect and see that it's refreshed. Now we click...  
Communication is taking place successfully.



Although here we saw an example of communication between web panels with components, the same is true for communicating multi-experience panels (for example, for native applications) with components.

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

# GeneXus™

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