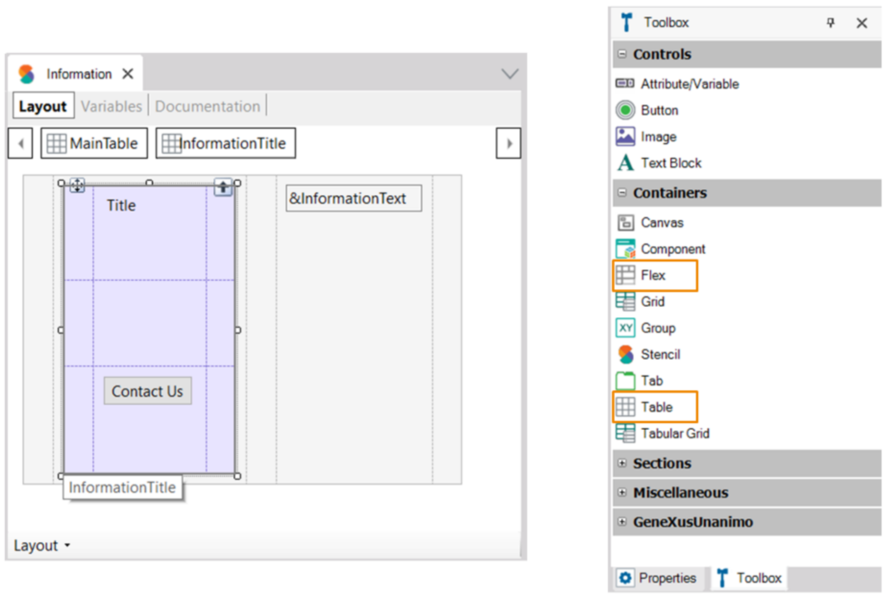
GeneXus by Globant

First Layout in GeneXus

Flex control instead of Table control

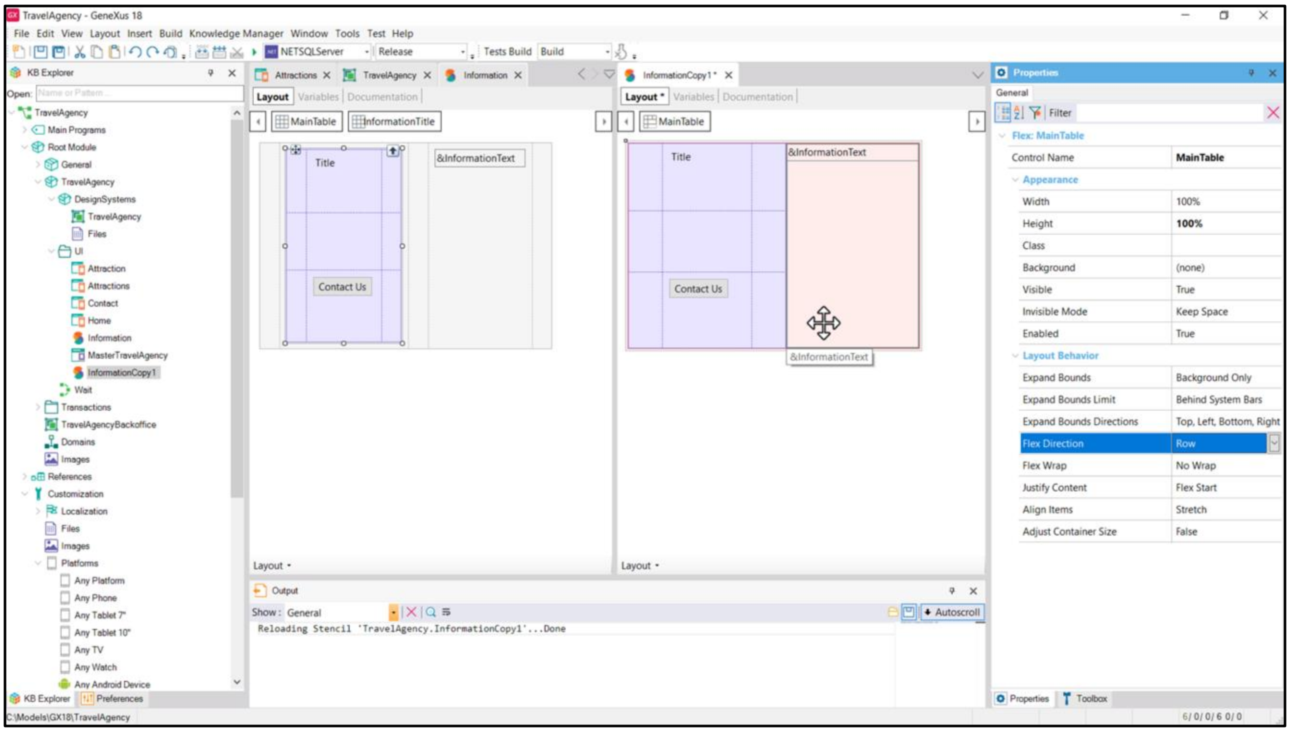
Cecilia Fernández

GeneXus by Globant



When we built the Stencil at the beginning of this module we chose to model it from the main table, and from another table for the textblock and the button.

But right there we said that we could also have chosen the Flex control instead of the Table control as the container, remember?



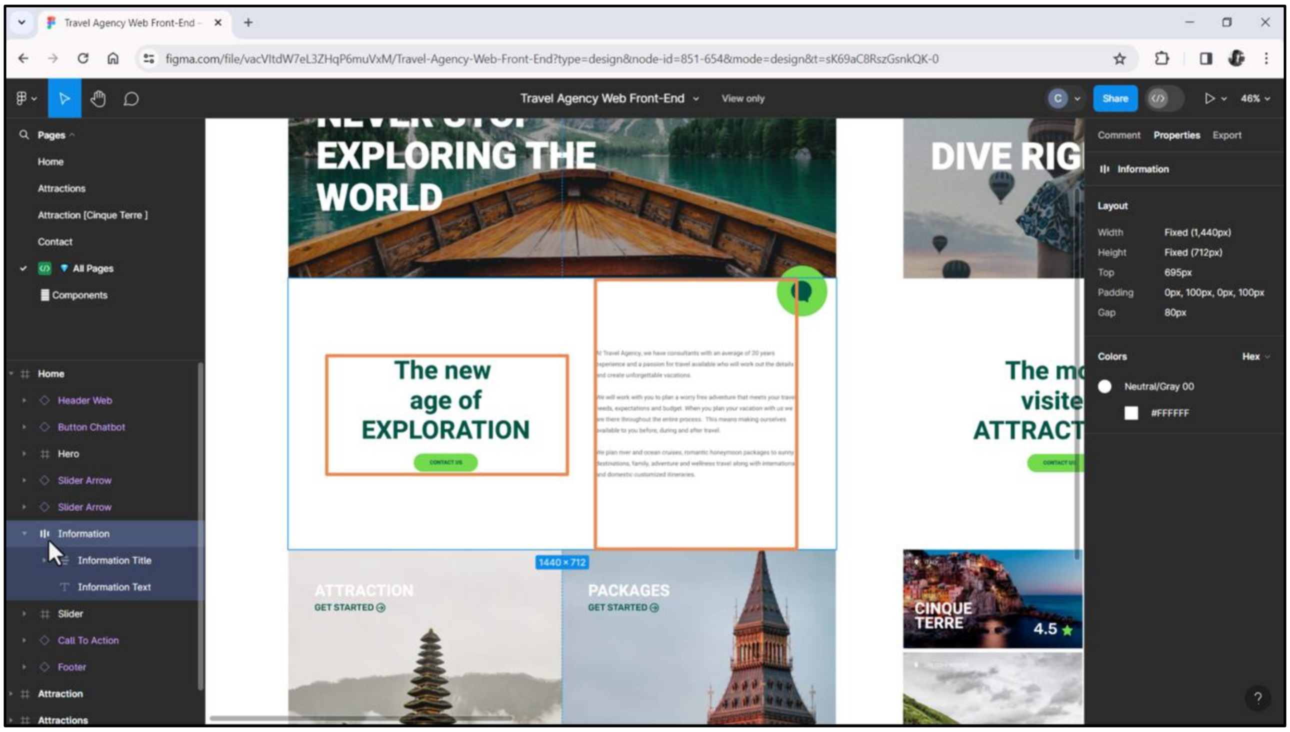
Flex: MainTable

Control Name	MainTable
Appearance	
Width	100%
Height	100%
Class	
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Limit	Behind System Bars
Expand Bounds Directions	Top, Left, Bottom, Right
Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Adjust Container Size	False

```
Reloading Stencil 'TravelAgency.InformationCopy1'...Done
```

What's more (I'm going to save our stencil with another name)... we had said that we could convert at any time a common table like this into Flex and vice versa. When we convert it we see that the rows and columns disappear, and these properties appear inside the Layout Behavior group. The first one that catches our attention is Flex Direction, whose default value is Row, and that will determine that the two elements contained by the flex are arranged horizontally along a single row, **without** columns.

It will be a little clearer in Figma.



Layout

Width	Fixed (1,440px)
Height	Fixed (712px)
Top	695px
Padding	0px, 100px, 0px, 100px
Gap	80px

Colors

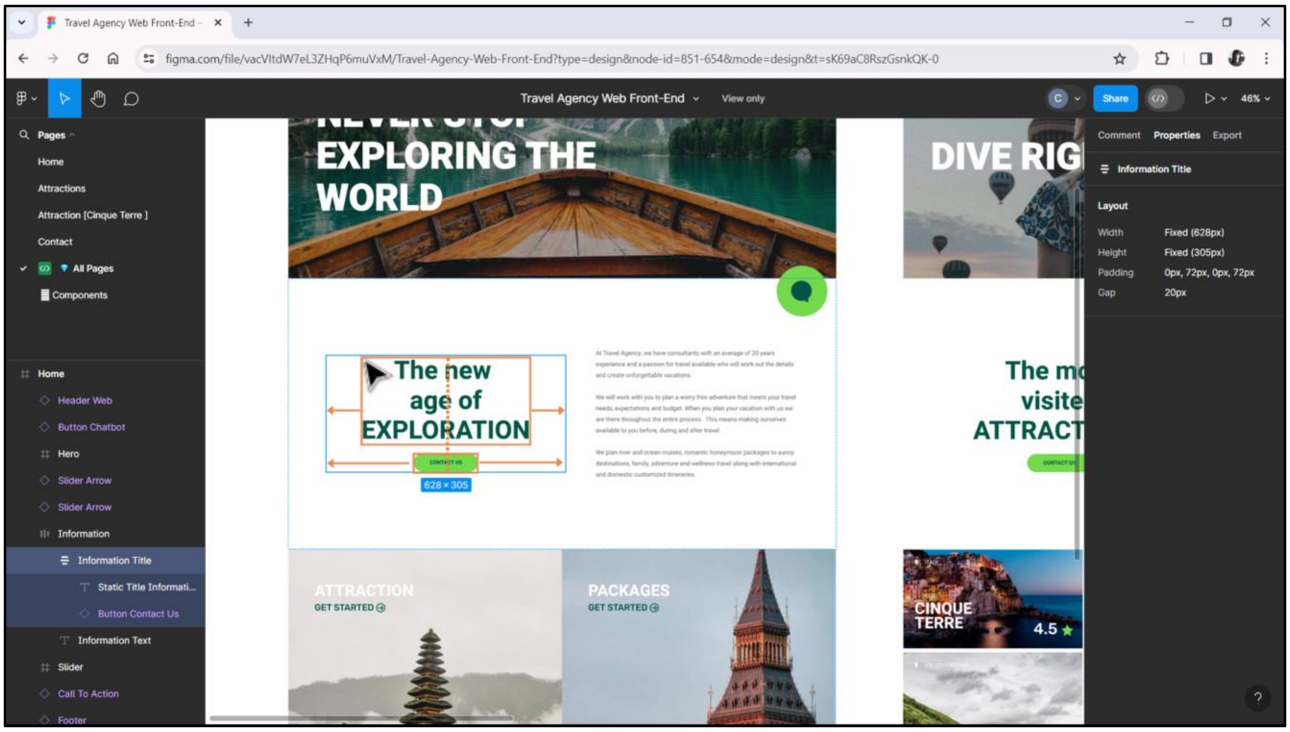
Neutral/Gray 00

#FFFFFF

This symbol appears for the Information frame and this other symbol for the frame containing the title and the button. These symbols tell us much more than it might seem.

The first one indicates that the elements of the frame (this and this) will be placed along a row, i.e. horizontally: first one, then the other. And also, that these elements can have different heights, but that they will be aligned in the center.

Actually, they have different heights: if we look at this one... it has this height, and if we look at this one, it has this other height, which is the height of the container. But they are both vertically aligned with respect to the center.

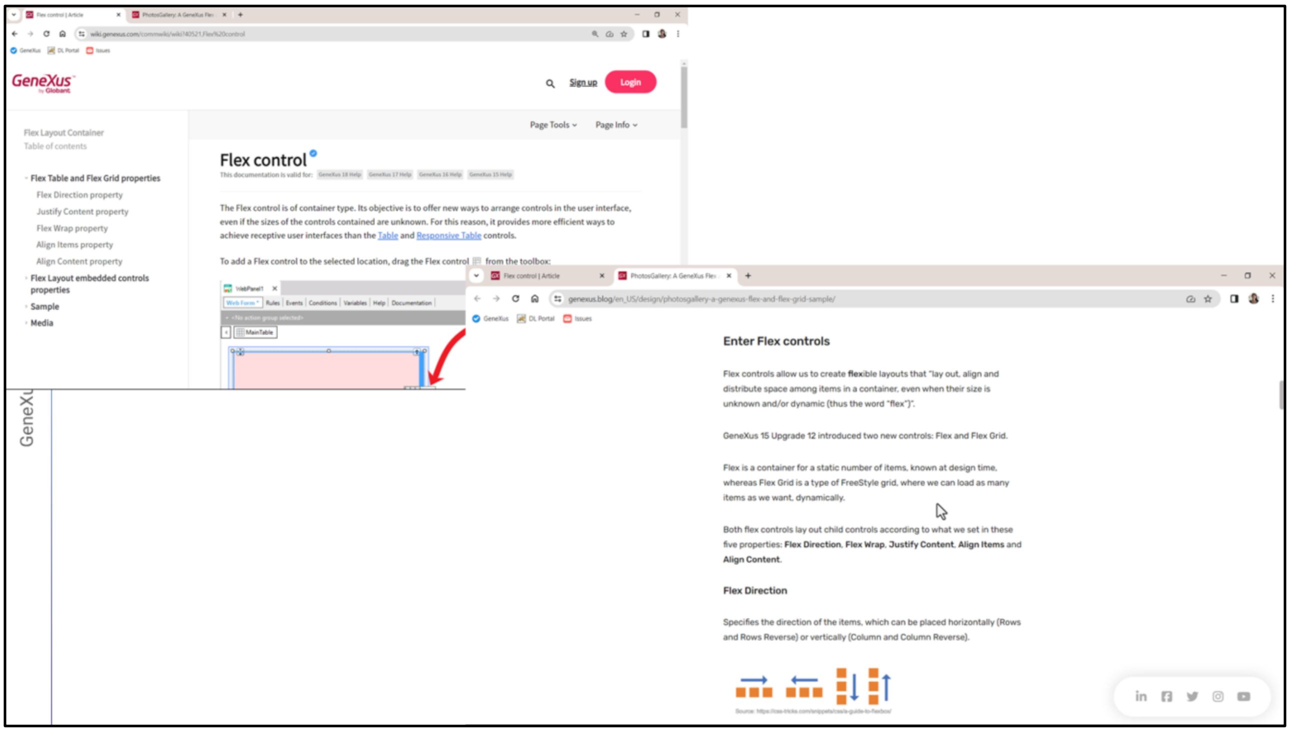


Width	Fixed (628px)
Height	Fixed (305px)
Padding	0px, 72px, 0px, 72px
Gap	20px

If we look at this other one, it is also a container with two elements, this time arranged around the column direction. First one, the text, and then the other, the button. And they are also center aligned with respect to the edges of the other direction. Just as these two, whose direction is row, are centered with respect to the vertical direction.

Chechu achieves this by saying that the container has what in Figma is known as Auto Layout. For us that is called Flex.

Basically, it means that the contained elements are going to be placed along one direction: either horizontally, like this one, or vertically, like this other one.



Flex Layout Container
Table of contents

- Flex Table and Flex Grid properties
  - Flex Direction property
  - Justify Content property
  - Flex Wrap property
  - Align Items property
  - Align Content property
- Flex Layout embedded controls properties
- Sample
- Media

Flex control

This documentation is valid for: GeneXus 18 Help GeneXus 17 Help GeneXus 16 Help GeneXus 15 Help

The Flex control is of container type. Its objective is to offer new ways to arrange controls in the user interface, even if the sizes of the controls contained are unknown. For this reason, it provides more efficient ways to achieve receptive user interfaces than the Table and Responsive Table controls.

To add a Flex control to the selected location, drag the Flex control from the toolbox:

WebPanel1
Web Form *
Rules
Events
Conditions
Variables
Help
Documentation
<No action group selected>
MainTable
GeneXu
Flex control | Article
PhotosGallery: A GeneXus Flex
genexus.blog/en_US/design/photosgallery-a-genexus-flex-and-flex-grid-sample/
GeneXus
DL Portal
Issues

Enter Flex controls

Flex controls allow us to create **flex**ible layouts that "lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word "flex")".

GeneXus 15 Upgrade 12 introduced two new controls: Flex and Flex Grid.

Flex is a container for a static number of items, known at design time, whereas Flex Grid is a type of FreeStyle grid, where we can load as many items as we want, dynamically.

Both flex controls lay out child controls according to what we set in these five properties: **Flex Direction**, **Flex Wrap**, **Justify Content**, **Align Items** and **Align Content**.

Flex Direction

Specifies the direction of the items, which can be placed horizontally (Rows and Rows Reverse) or vertically (Column and Column Reverse).

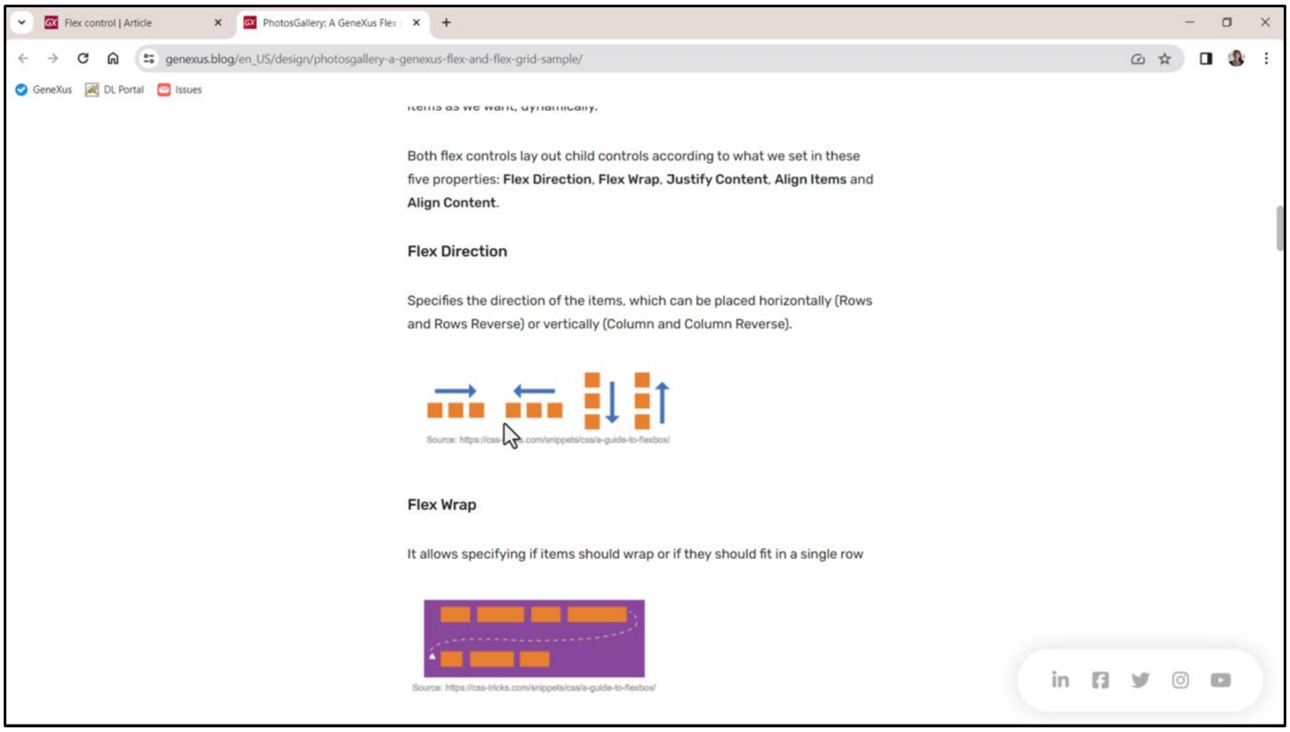
Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

This GeneXus wiki page explains the control. And at the end there is a link to this other page that gives an example.

I will use this page to see it graphically and make it clearer.

It says here that these containers allow you to create **flexible** layouts, which allow you to place items, align them and distribute the space between them in a container, of course, even (or especially, I would say) when their size is unknown beforehand or is dynamic.

A series of properties are then displayed to configure the direction and layout of the controls.



Flex control | Article
PhotosGallery: A GeneXus Flex
genexus.blog/en_US/design/photosgallery-a-genexus-flex-and-flex-grid-sample/
GeneXus DL Portal Issues

items as we want, dynamically.

Both flex controls lay out child controls according to what we set in these five properties: **Flex Direction**, **Flex Wrap**, **Justify Content**, **Align Items** and **Align Content**.

Flex Direction

Specifies the direction of the items, which can be placed horizontally (Rows and Rows Reverse) or vertically (Column and Column Reverse).

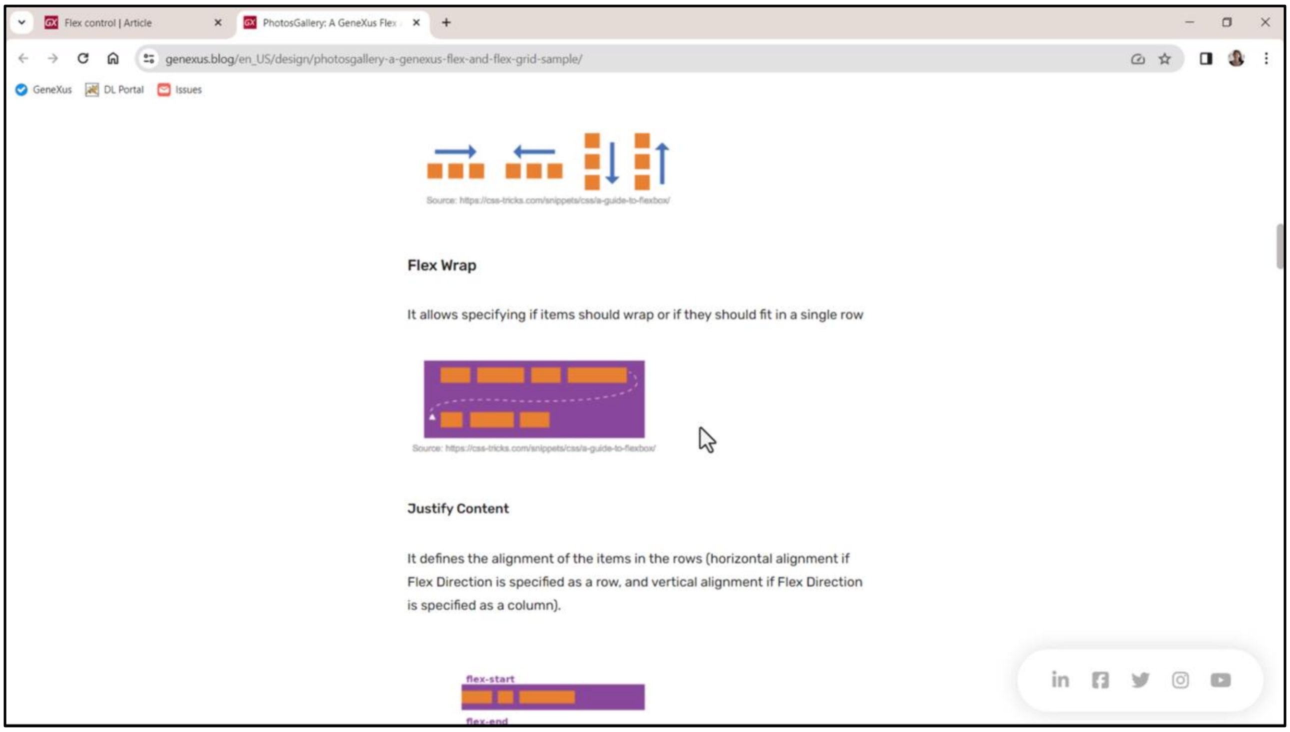
Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Flex Wrap

It allows specifying if items should wrap or if they should fit in a single row

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

This would be the case of one container in row direction, this one in row reverse, this one in column direction, and this one in column reverse. In the example, each container has 3 items.


Flex control | Article
PhotosGallery: A GeneXus Flex
genexus.blog/en_US/design/photosgallery-a-genexus-flex-and-flex-grid-sample/
GeneXus
DL Portal
Issues

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Flex Wrap

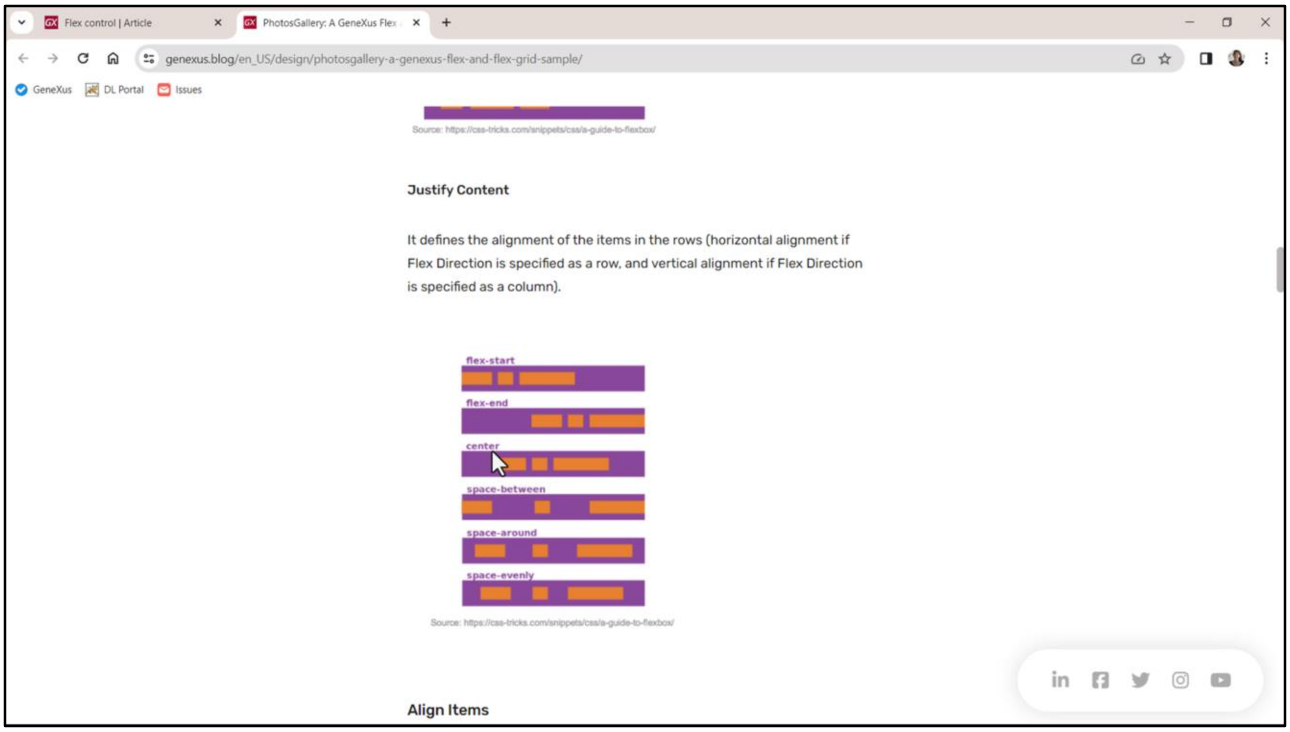
It allows specifying if items should wrap or if they should fit in a single row

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Justify Content

It defines the alignment of the items in the rows (horizontal alignment if Flex Direction is specified as a row, and vertical alignment if Flex Direction is specified as a column).

In this other example, we decide whether to allow wrapping if the size of the items, that is, their width, added together, exceeds the limits of the container. So in this case we are allowing the wrap, that is, to extend in a line below, given that the direction is row, of course. If the direction were a column, it would extend into a second column.



Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

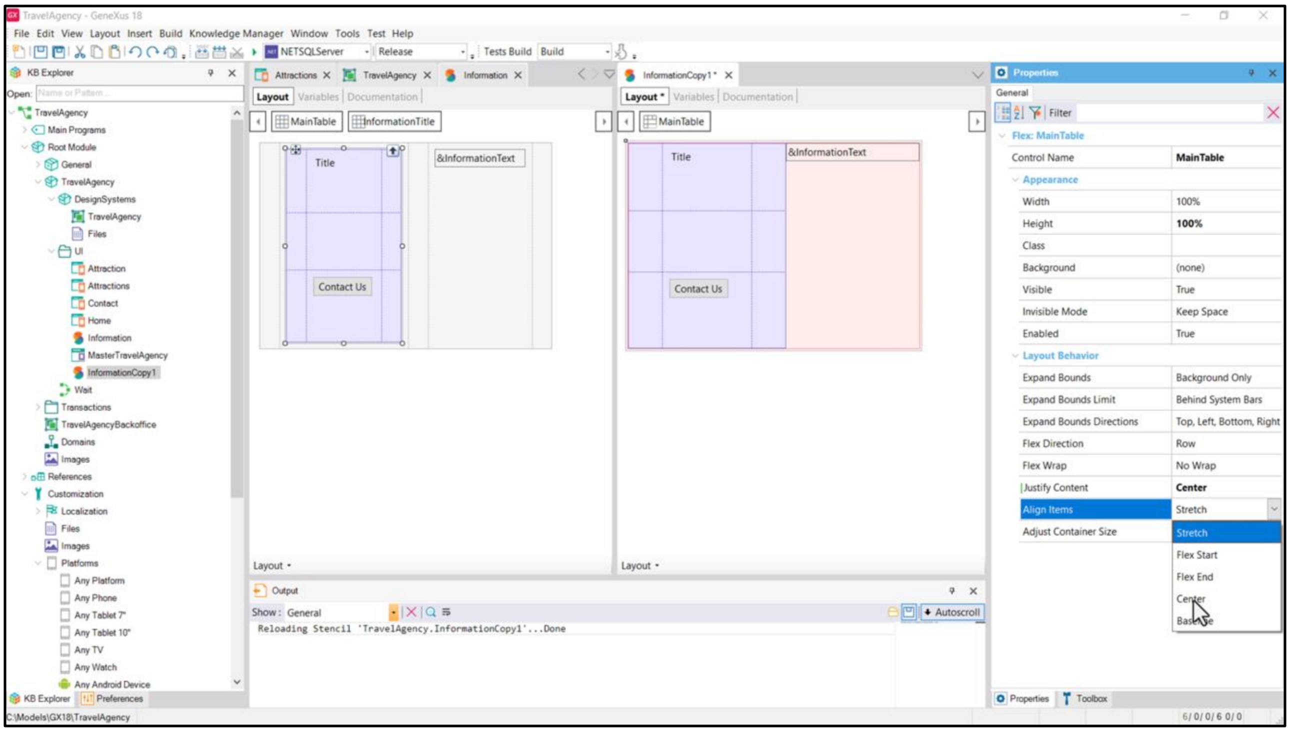
Justify Content

It defines the alignment of the items in the rows (horizontal alignment if Flex Direction is specified as a row, and vertical alignment if Flex Direction is specified as a column).

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Align Items

This other property indicates how we want the content to be justified. For example, if we want the items to be centered with respect to the direction, which in this case is row. This would be precisely the case of our Information flex.



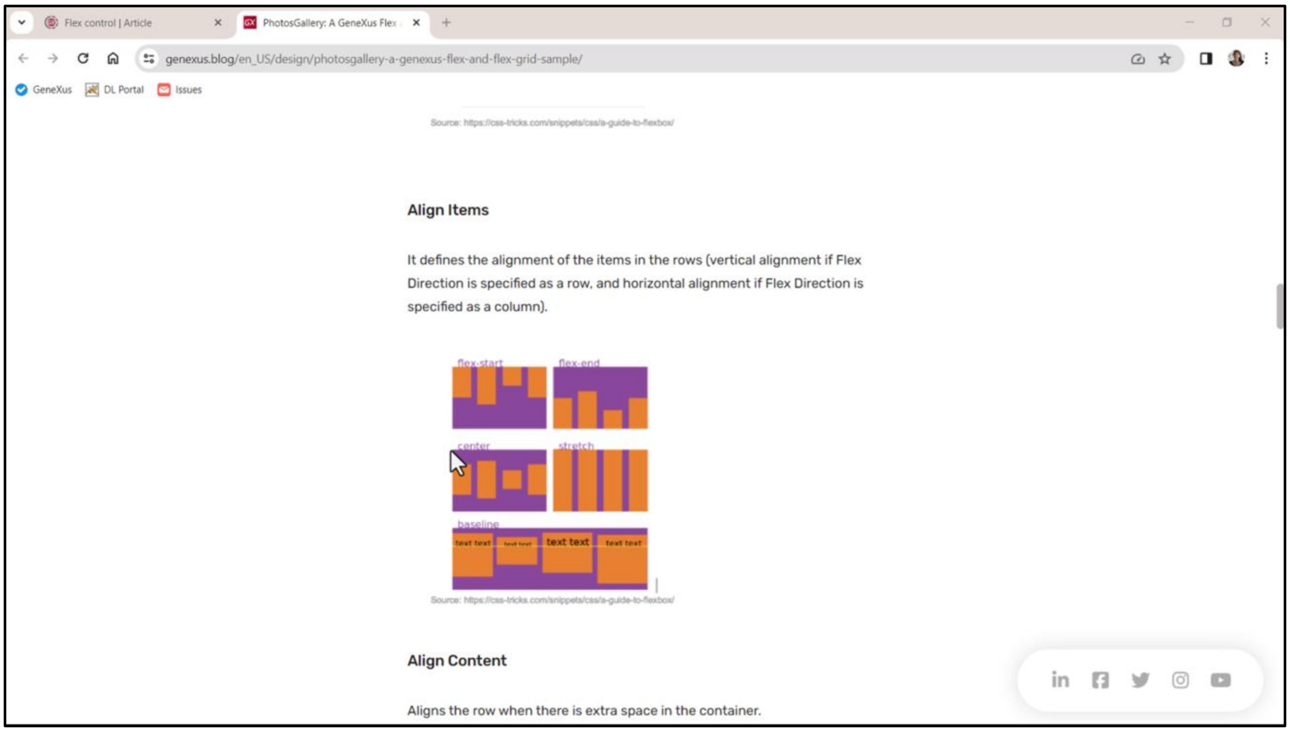
Flex: MainTable

Control Name	MainTable
Appearance	
Width	100%
Height	100%
Class	
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Limit	Behind System Bars
Expand Bounds Directions	Top, Left, Bottom, Right
Flex Direction	Row
Flex Wrap	No Wrap
Justify Content	Center
Align Items	Stretch
Adjust Container Size	

```
Reloading Stencil 'TravelAgency.InformationCopy1'...Done
```

So we come here, and we are going to do... Flex Direction “Row”. For Flex Wrap we are going to leave “No Wrap” at first. And “Center” justification (we see the options).

We'll set the alignment of the items with respect to the top and bottom to be centered as well.



Flex control | Article

PhotosGallery: A GeneXus Flex

genexus.blog/en_US/design/photosgallery-a-genexus-flex-and-flex-grid-sample/

GeneXus DL Portal Issues

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Align Items

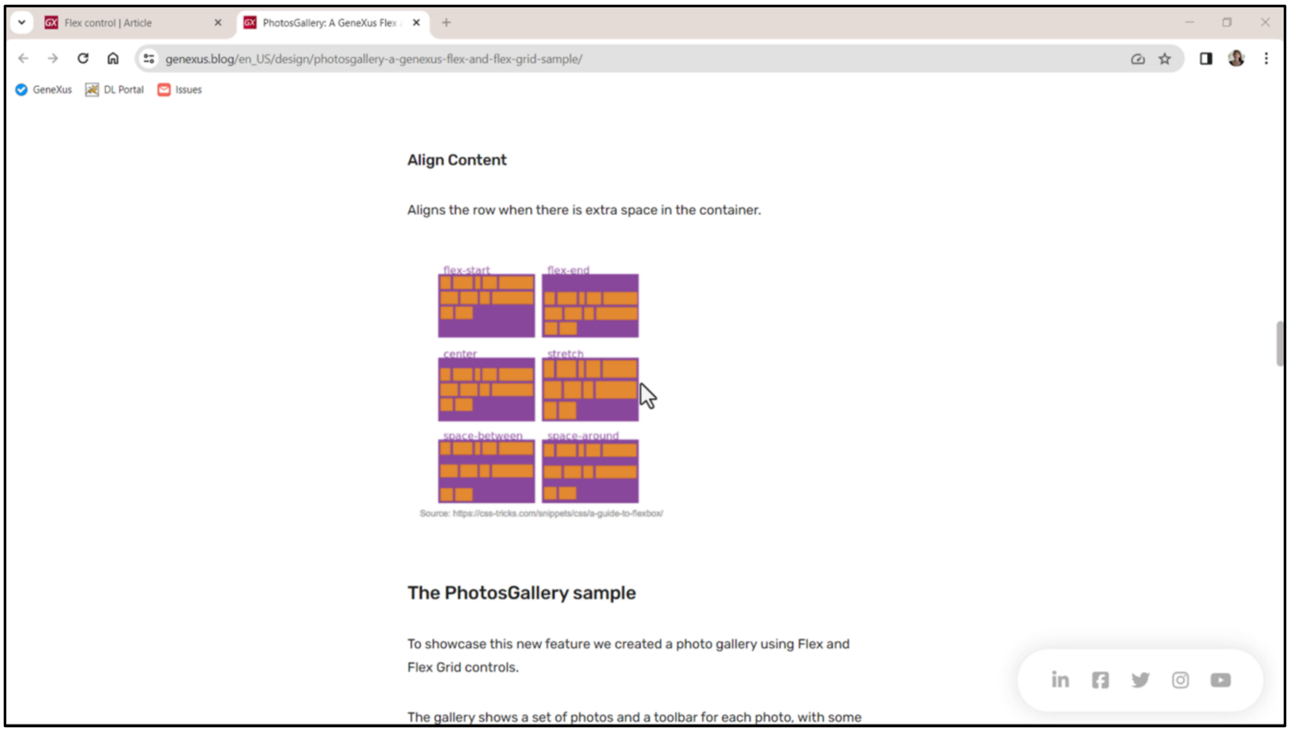
It defines the alignment of the items in the rows (vertical alignment if Flex Direction is specified as a row, and horizontal alignment if Flex Direction is specified as a column).

Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Align Content

Aligns the row when there is extra space in the container.

And here it will be clear what we are saying with that.



Flex control | Article

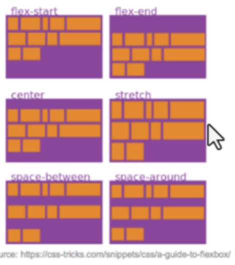
PhotosGallery: A GeneXus Flex

genexus.blog/en_US/design/photosgallery-a-genexus-flex-and-flex-grid-sample/

GeneXus DL Portal Issues

Align Content

Aligns the row when there is extra space in the container.

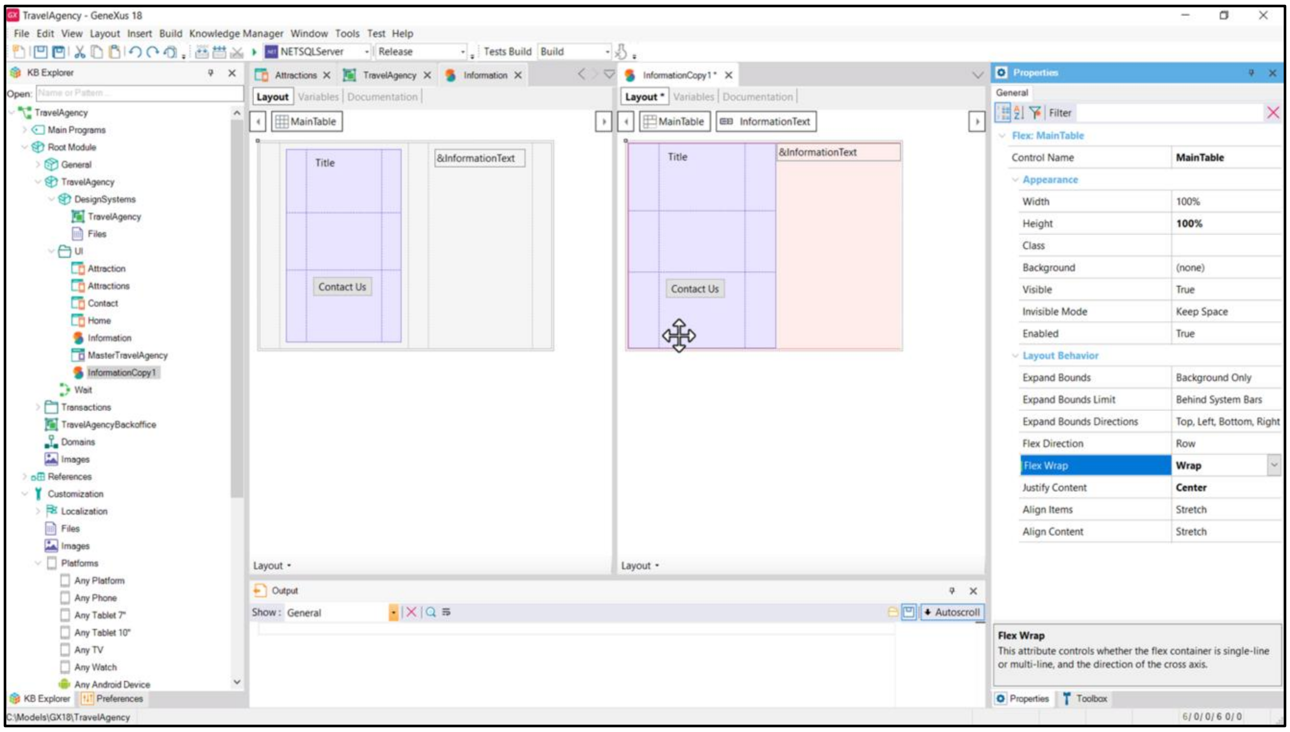

flex-start
flex-end
center
stretch
space-between
space-around
Source: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

The PhotosGallery sample

To showcase this new feature we created a photo gallery using Flex and Flex Grid controls.

The gallery shows a set of photos and a toolbar for each photo, with some

On the other hand, if we allow the wrap, then with this property we determine how to adjust the content with respect to the excess space.



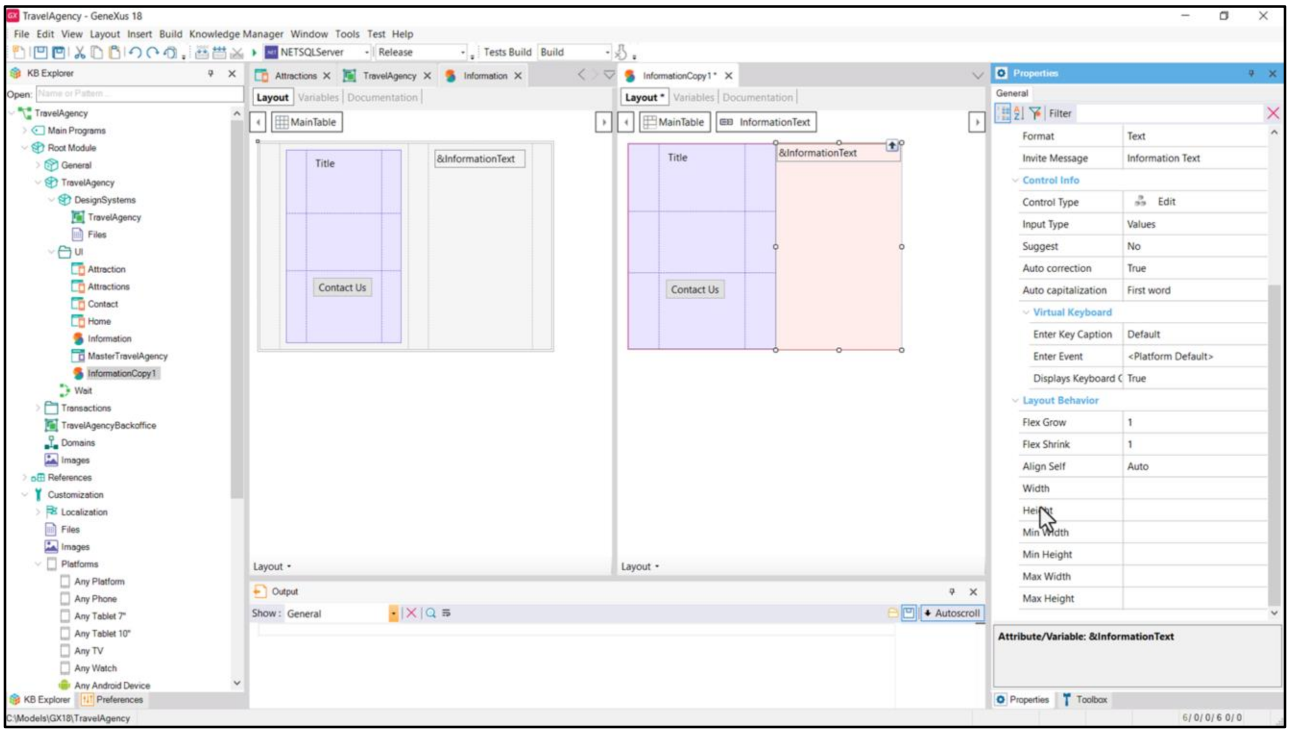
Flex: MainTable	
Control Name	MainTable
Appearance	
Width	100%
Height	100%
Class	
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Limit	Behind System Bars
Expand Bounds Directions	Top, Left, Bottom, Right
Flex Direction	Row
Flex Wrap	Wrap
Justify Content	Center
Align Items	Stretch
Align Content	Stretch

Flex Wrap
This attribute controls whether the flex container is single-line or multi-line, and the direction of the cross axis.

Well, I didn't intend to tell you all the possibilities of the control, but simply to introduce it because this control is widely used both on the web and in native applications, precisely because of its flexibility.

In this case, for example, this control would be more useful than the Table control if we wanted –when the screen width decreases if the two elements don't fit that width– to have them displayed in two rows (i.e. with Wrap). The table doesn't allow this, it is much more structured: it has rows and columns. And in fixed quantities.

Here we only have a row (if the direction is row, like this one) without columns, that is to say, where the elements are placed one next to the other according to the available space. And if they run out of space, if the Wrap property is turned on, then a second row is created and so on until completing all the items.



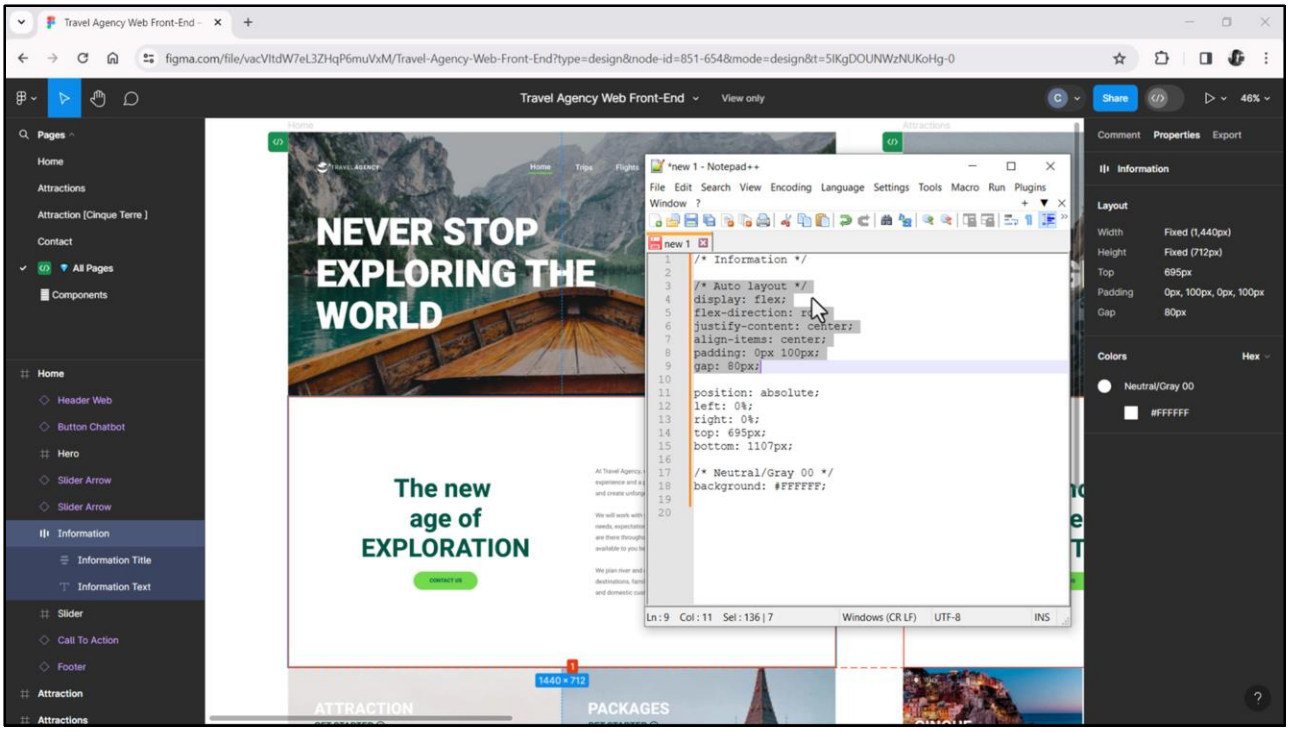
Properties — General

Format	Text
Invite Message	Information Text
Control Info	
Control Type	Edit
Input Type	Values
Suggest	No
Auto correction	True
Auto capitalization	First word
Virtual Keyboard	
Enter Key Caption	Default
Enter Event	<Platform Default>
Displays Keyboard C	True
Layout Behavior	
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	
Min Height	
Max Width	
Max Height	

Attribute/Variable: &InformationText

When a control is inside a flex container, it takes properties, these here, precisely because of that situation.

Of course, as with any other control, we can define design properties for the Flex control, all those that we saw statically, or the more well-known ones of margin, padding, etc., through classes.



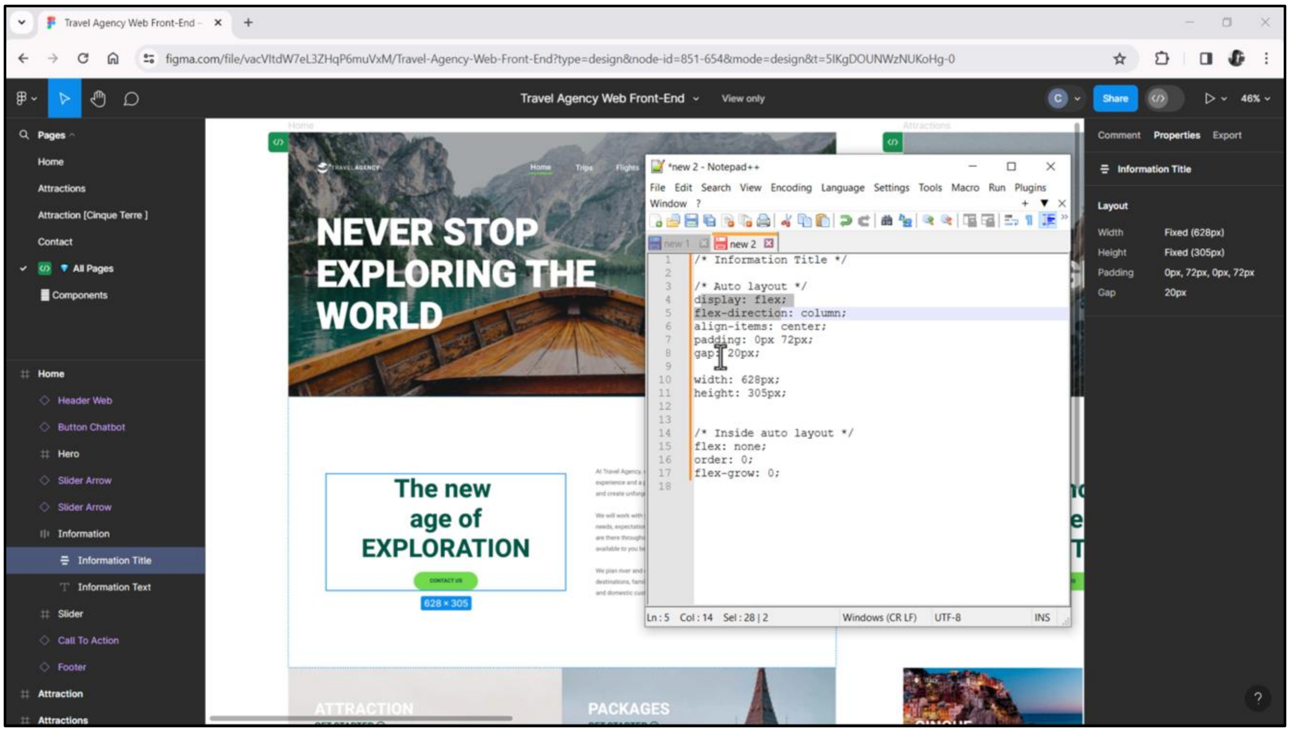
```
/* Information */

/* Auto layout */
display: flex;
flex-direction: ro
justify-content: center;
align-items: center;
padding: 0px 100px;
gap: 80px;

position: absolute;
left: 0%;
right: 0%;
top: 695px;
bottom: 1107px;

/* Neutral/Gray 00 */
background: #FFFFFF;
```

Layout
Width Fixed (1,440px)
Height Fixed (712px)
Top 695px
Padding 0px, 100px, 0px, 100px
Gap 80px

Colors
Neutral/Gray 00
#FFFFFF

For example, going back to Figma, when Chechu defined this container with Auto Layout, she set this horizontal direction, and paddings appeared to define the four edges: top, right, bottom, left. And it was also possible to define the Gap, which is the horizontal separation between the elements, which she had set to 80px.

Look at the CSS code that appears when we ask for this...
Doesn't all this sound familiar? Display flex, flex-direction row, justify-content center, align-item center. We statically set these properties at the level of our Flex control, and these two, padding and gap, we could place them in a class that we will associate later to our Flex control. Or we could also place them all within that class, because all that we can define at the level of the control can also be defined at the level of a class properties, right?



```
/* Information Title */

/* Auto layout */
display: flex;
flex-direction: column;
align-items: center;
padding: 0px 72px;
gap: 20px;

width: 628px;
height: 305px;


/* Inside auto layout */
flex: none;
order: 0;
flex-grow: 0;
```

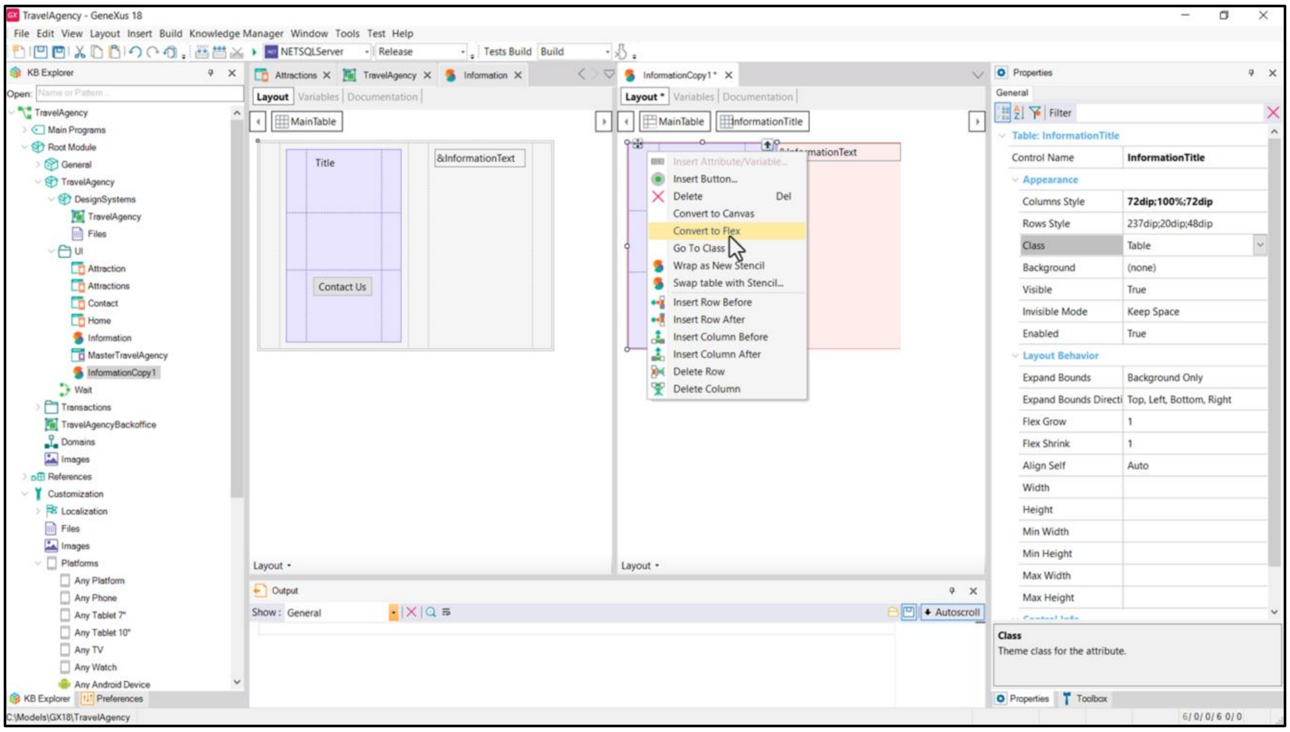
Layout

Width	Fixed (628px)
Height	Fixed (305px)
Padding	0px, 72px, 0px, 72px
Gap	20px

Well, in the same way, we see that this other container also has Auto Layout, but in the other direction, in the column direction.

Here we see the Padding and Gap properties.

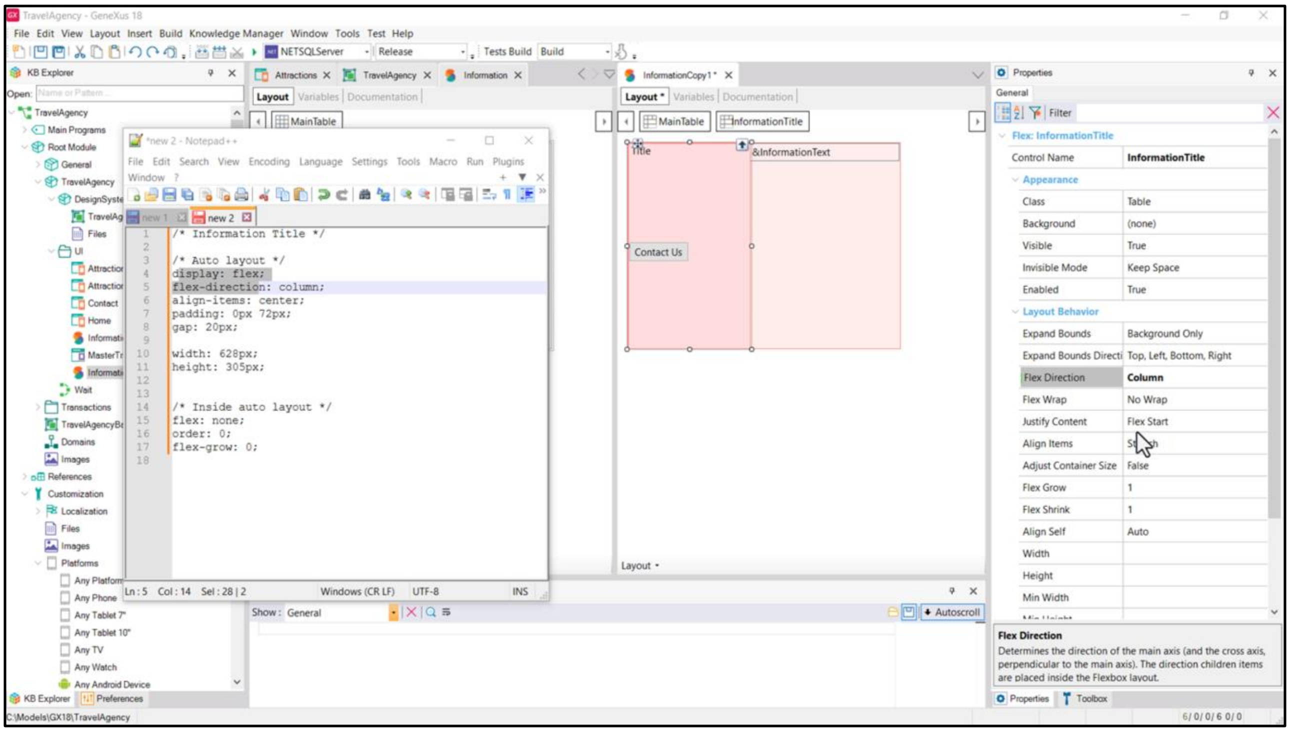
And again, if we copy the CSS properties, here is the display, the direction: column this time, align-items and the properties Padding and Gap.



Control Name	InformationTitle
Columns Style	72dip;100%;72dip
Rows Style	237dip;20dip;48dip
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	
Min Height	
Max Width	
Max Height	

Class
Theme class for the attribute.

So we could also define this table as Flex.



```
/* Information Title */

/* Auto layout */
display: flex;
flex-direction: column;
align-items: center;
padding: 0px 72px;
gap: 20px;

width: 628px;
height: 305px;


/* Inside auto layout */
flex: none;
order: 0;
flex-grow: 0;
```

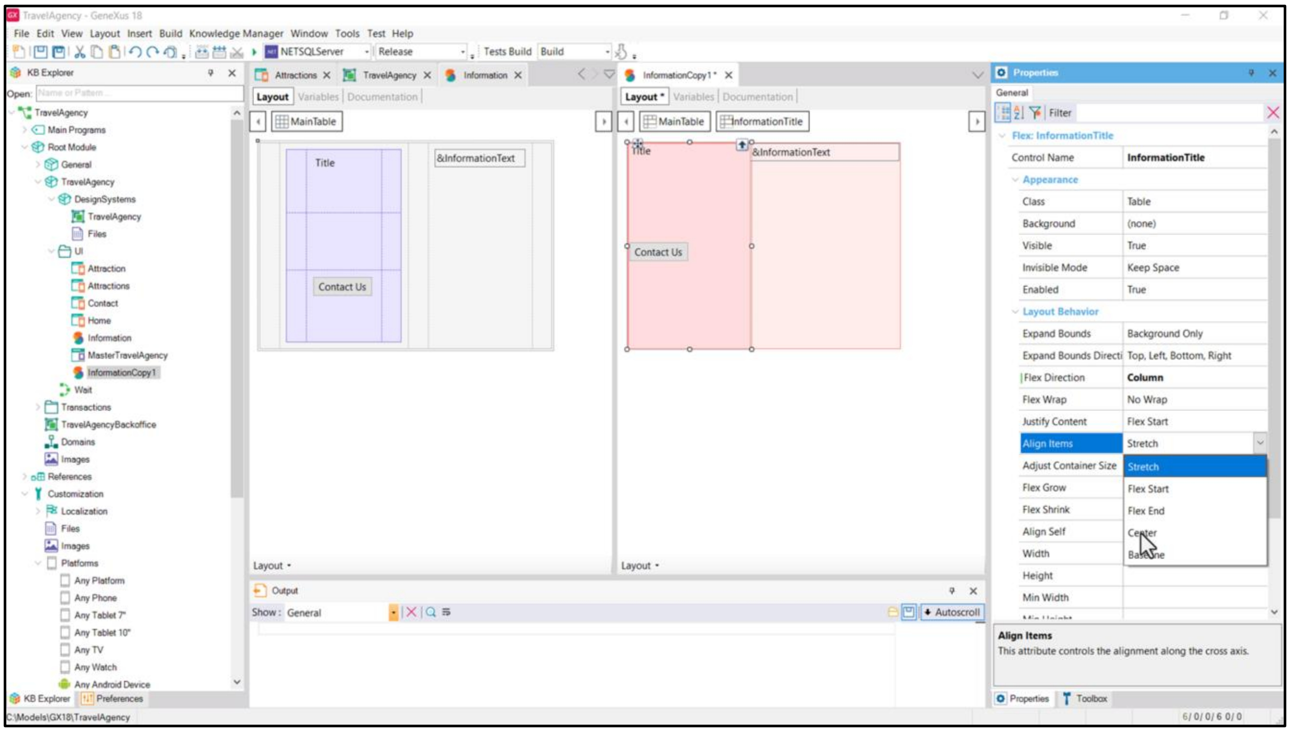
Flex: InformationTitle

Control Name	InformationTitle
Appearance	
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Direction	Column
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Adjust Container Size	False
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	

Flex Direction
Determines the direction of the main axis (and the cross axis, perpendicular to the main axis). The direction children items are placed inside the Flexbox layout.

And set, of course, the column direction so that it is displayed in this way.

From Figma's CSS we also extracted this, that align-items is centered, which represents these items in relation to the edges of the other direction, right?



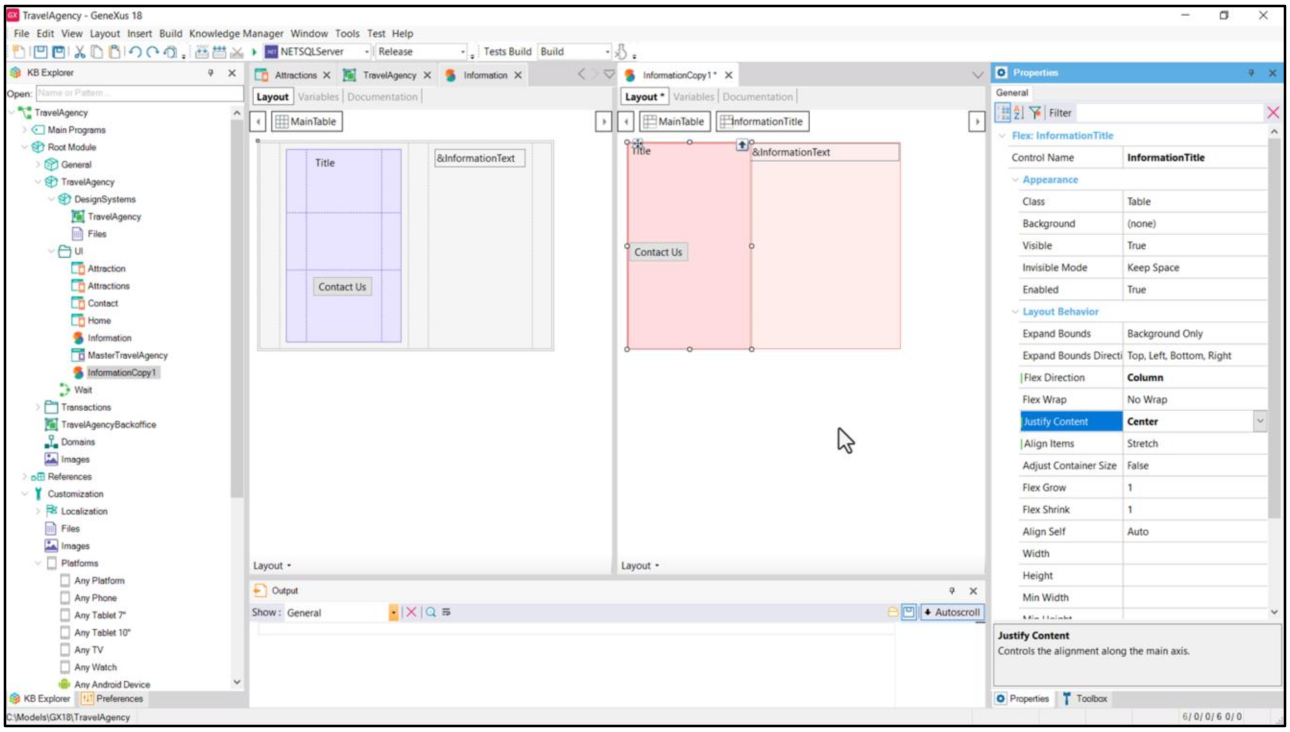
Flex: InformationTitle

Control Name	InformationTitle
Appearance	
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Direction	Column
Flex Wrap	No Wrap
Justify Content	Flex Start
Align Items	Stretch
Adjust Container Size	
Flex Grow	
Flex Shrink	
Align Self	
Width	
Height	
Min Width	

Stretch
Flex Start
Flex End
Center
Baseline

Align Items
This attribute controls the alignment along the cross axis.

And here we would have to select Center. Well, they are disappearing, this should not be happening... I'm going to leave the default, we'll see it later.



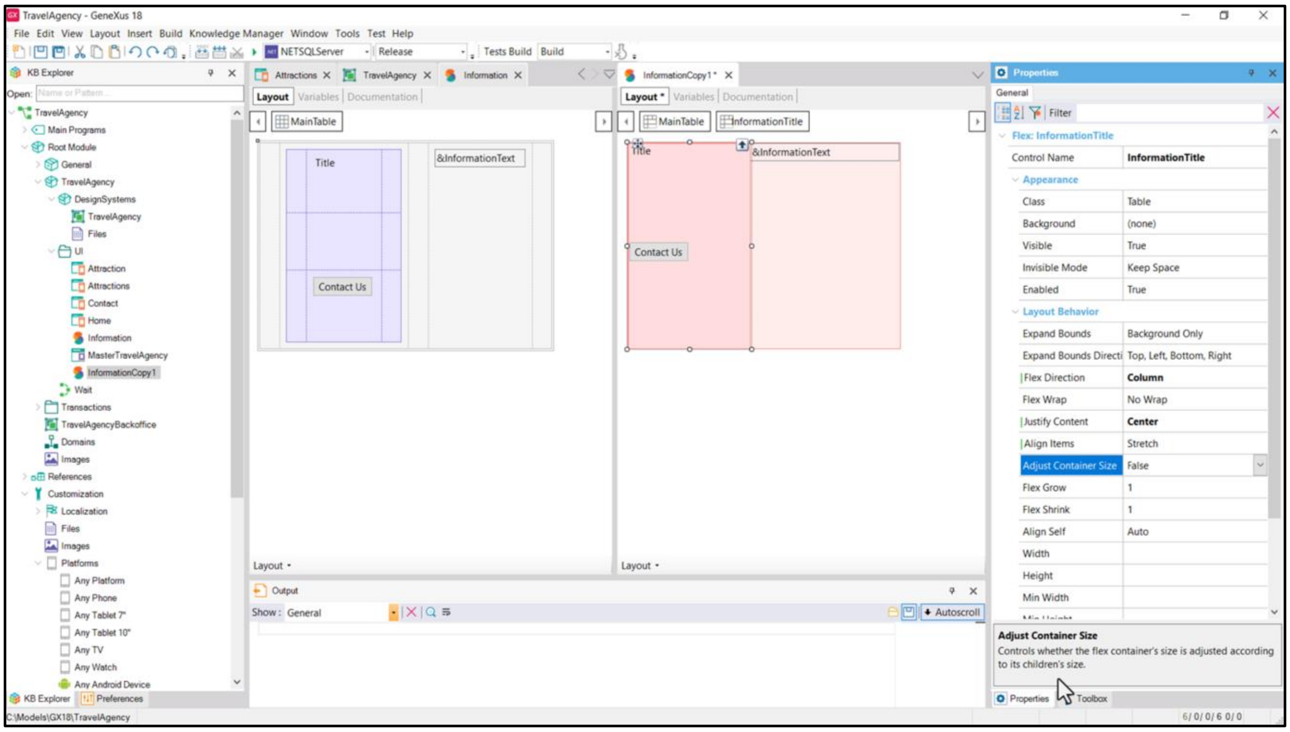
Properties

Flex: InformationTitle

Control Name	InformationTitle
Appearance	
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Direction	Column
Flex Wrap	No Wrap
Justify Content	Center
Align Items	Stretch
Adjust Container Size	False
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	

Justify Content
Controls the alignment along the main axis.

The content justification does have to do with the same direction of the flex, doesn't it? In this case it is column, so we would also have to say centered for our case, even if the CSS didn't indicate it.



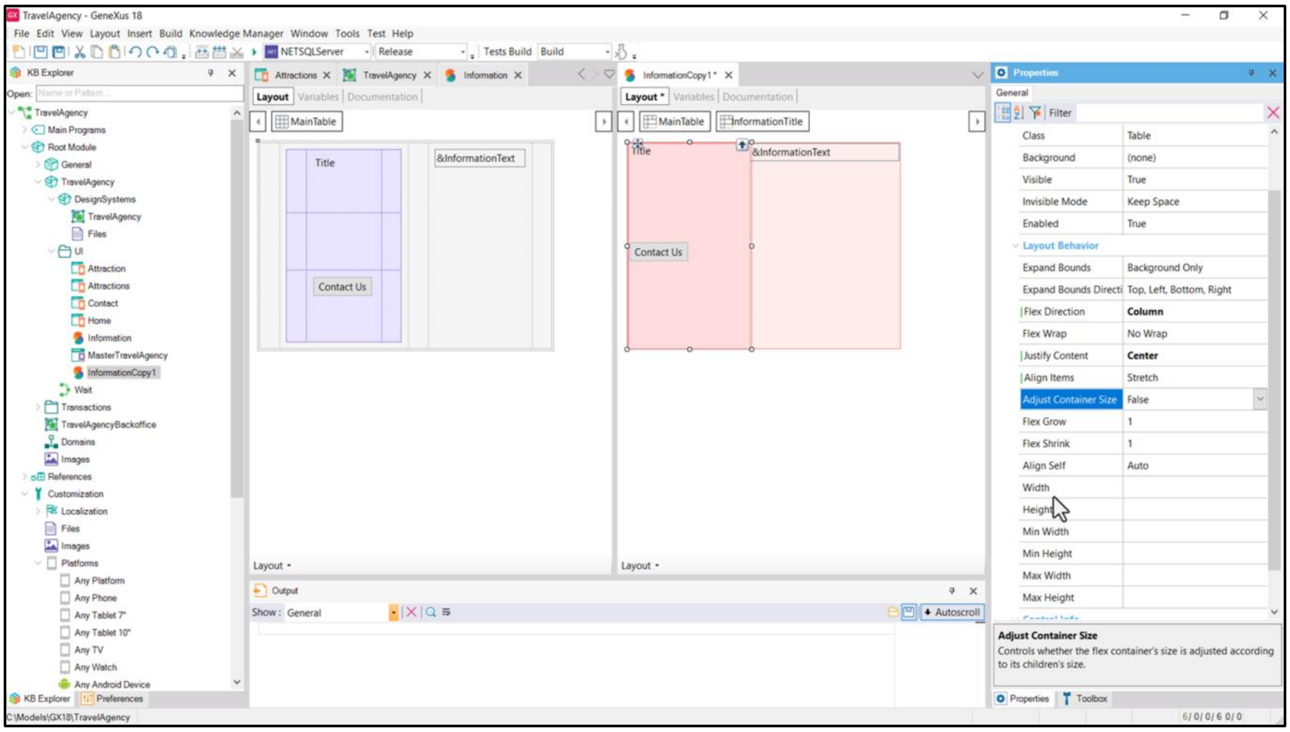
Flex: InformationTitle

Control Name	**InformationTitle**
Appearance	
Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Direction	**Column**
Flex Wrap	No Wrap
Justify Content	**Center**
Align Items	Stretch
Adjust Container Size	False
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	

Adjust Container Size
Controls whether the flex container's size is adjusted according to its children's size.

Since it doesn't have wrapping –we didn't set it to have Wrap– then this property comes into play and controls whether the size of the container is going to be adjusted according to the size of the children.

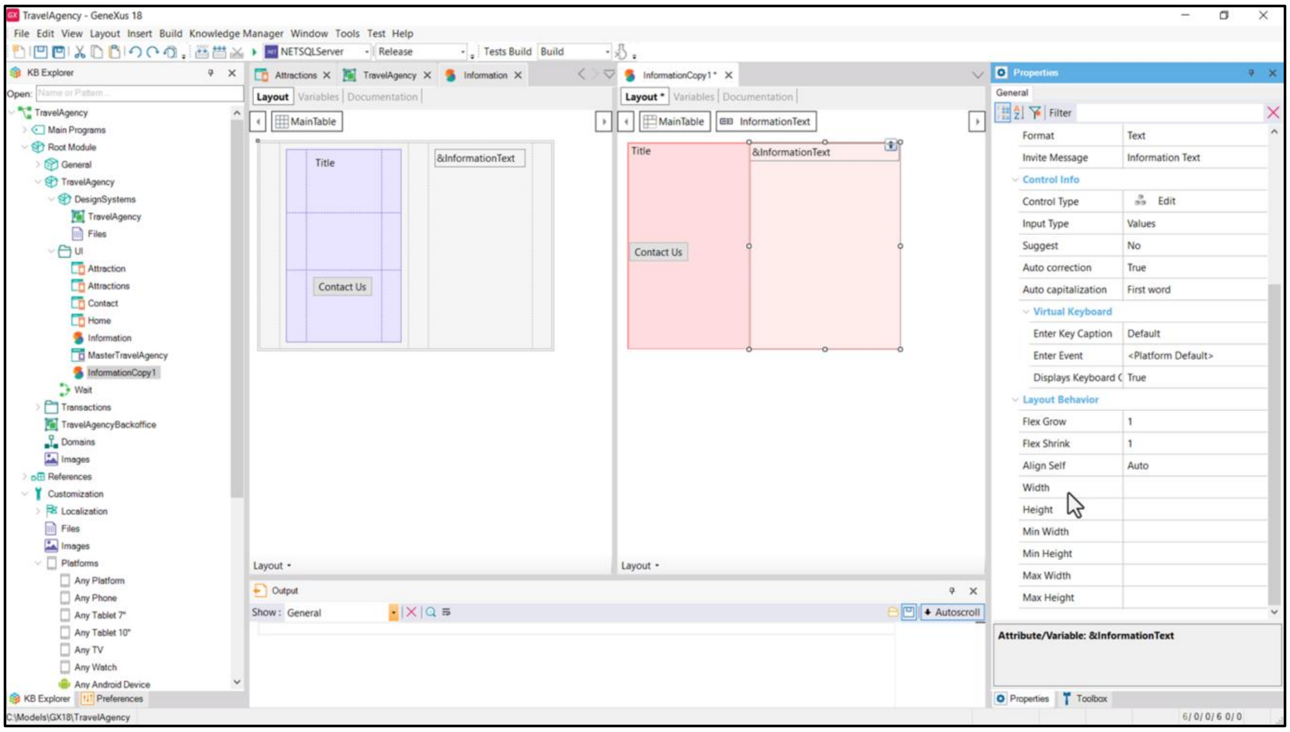
Well, we would have to work with these aspects in order to place our elements in the correct way according to Chechu's design. At first it may seem simpler to use the flex containers because we can copy Chechu's design criteria directly... with Auto Layout we can transfer it almost automatically to GeneXus. But we have to be careful because this is a bit trickier...



Class	Table
Background	(none)
Visible	True
Invisible Mode	Keep Space
Enabled	True
Layout Behavior	
Expand Bounds	Background Only
Expand Bounds Directi	Top, Left, Bottom, Right
Flex Direction	**Column**
Flex Wrap	No Wrap
Justify Content	**Center**
Align Items	Stretch
Adjust Container Size	False
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	
Min Height	
Max Width	
Max Height	

Adjust Container Size
Controls whether the flex container's size is adjusted according to its children's size.

...note all the properties that are appearing now within this group... in this case more appear because there are also those related to the participation of this internal Flex within this other Flex.



Format	Text
Invite Message	Information Text
Control Info	
Control Type	Edit
Input Type	Values
Suggest	No
Auto correction	True
Auto capitalization	First word
Virtual Keyboard	
Enter Key Caption	Default
Enter Event	<Platform Default>
Displays Keyboard (	True
Layout Behavior	
Flex Grow	1
Flex Shrink	1
Align Self	Auto
Width	
Height	
Min Width	
Min Height	
Max Width	
Max Height	

Attribute/Variable: &InformationText

As I had shown you before, all these properties appeared here, such as size, width and height, which had to do with this control's role within the parent Flex.

Well, because of these somewhat trickier issues I decided to start teaching you with Table and not with Flex. Flex may seem a bit more challenging because of those properties that have to be better understood. Sometimes it gets confusing; for example, if I'm talking about content justification, what is its direction? If I have the alignment of the item... Then one starts to get confused, but when we get the hang of it, it's quite simple. And well, it actually has to be used carefully.

When to use Flex? When I'm implementing a design where the elements that will be for example along a row (if that is the direction) are not so structurally fixed in columns, and certainly, of course, when we want the wrap functionality.

In this case, we didn't need it at first because as we said when the screen width doesn't fit both elements another different design will be made, which we haven't seen yet because I asked Chechu to design them in another file, so as not to get confused now.

OK, I'll stop here as it has been introduced, which is what I wanted, and later on we'll see if we need to use a Flex container instead of a Table. By later on I mean when we continue developing the screens of our application.

A grid can also be flex. I mention this now and I will mention it again when we implement one of the grids in the future.

I will end this video now because I want to move on to the next module but you can, if you want, try what I didn't try. For example, assign classes to the Flex with the corresponding properties, and pay attention to the sizes given to the controls, explore a little and try to make the application with the stencil with Flex look the same as it did in the other way, as we had it done. That's the challenge I leave for you.

See you in the next module.