Flex in a Week, Flex 4.5
Video 4.08: Laying out an application

In this training, you have used both Flex containers and controls.

In this video, you will learn about the parent/child relationship between these components and how to define the position and size of child components explicitly and with percentage values.

Every visual element in a Flex application is part of a hierarchy called the display list.

At the top of the MXML application hierarchy, you will find the Application container.

In other words, it is the ultimate parent of the application.

A parent component is always a container of some sort that can hold other containers or controls.

A child component, therefore, can be either a container or a control.

When laying out and sizing a component in the application, you must be aware of properties set on the parent, the component itself and its siblings.

All of these factors can affect how a component is displayed.

If you are struggling with laying out a particular component, it's helpful to remember that a child is constrained to its immediate parent.

In other words, a Button control inside of a Panel container can never be positioned outside of the Panel container.

Always look to the control's relationship to its immediate parent container first, when trying to debug a layout issue.

This is the starter application for this video and associated exercise.

This is the login page for the application.

When I submit the form – which you will learn how to authenticate in a future video - there is a header area with title text and a Logout button.

The four areas for Employee of the Month, Employee Directory, Cafeteria Special and Monthly Events are panel containers.

Over the course of the rest of this series, I will gradually enhance this application display.

When you layout components in containers, you can do so in three ways.

Absolute positioning uses the BasicLayout class and requires that you explicitly declare x and y property values for all of the children if you do not want them all piled up on top of each other in the upper left corner of the container.

Relative positioning uses the VerticalLayout, HorizontalLayout or the TileLayout classes and all of the children are laid out relative to one another vertically stacked, horizontally placed or tiled in rows and columns.

Constraint-based layout also positions the children with the BasicLayout class, but does not define x and y properties.

Instead, the children define their position relative to their parent container using anchor locations and pixel values.

You will learn about constraint-based layout in the next video.

In the main application code, you can see that the layout property is set to BasicLayout, which means that all of the immediate child elements will be laid out absolutely, using x and y property values.

I am locating the EmployeeOfTheMonth Panel container using the Outline view.

You can see that the layout property is set to the VerticalLayout class.

This means that its immediate children are stacked vertically relative to one another and the x and y properties are ignored.

I can use Outline view to also verify that the Employee Directory, Cafeteria Special and Monthly Events panels are also laid out in the VerticalLayout orientation.

When sizing components, you can do so in four ways.

The first is default sizing.

Some, but not all components have an explicit or relative default size.

For instance, a Spark ComboBox control, is by default 146 pixels wide by 23 pixels tall while a Spark Button control sizes itself to fit the text displayed in its label property.

You can also define absolute sizes to your components by setting width and height properties to explicit pixel dimensions.

The components will stay that size regardless of the size of the parent container.

You can define percentage sizing by setting percentage values in the width and height properties of a child component.

This is a relative value to the parent container.

The child component's size will change if the parent's size changes and will be the specified percentage of the parent size.

Lastly, you can define constraint-based sizing.

As I mentioned before, constraint-based layouts define a child's display relative to pixel offsets from anchor locations within the parent container.

Like percentage sizing, constraint-based sizing can change a child's size based on the parent's size change.

You will learn more about constraint-based sizing in the next video.

Back in the browser, look at the Submit and Desktop Delivery buttons. You can see that the Button controls of the application size to fit the text displayed in its label.

I am return to Flash Builder.

Look at the properties for each of the Panel containers. Each panel uses absolute positioning with set x and y values and they also use absolute sizing with specific width and height values.

Because these panels have explicit x,y, width, and height values, the containers will not move and will stay the same size.

If I scale the browser to the right, you can see that the three columns of content don't take advantage of this additional space.

They are all statically positioned in the application.

I am going to apply percentage values to three of the Panel containers to have them scale appropriately to the available space.

Each of the containers currently defines an x and y property.

Back in the main application file, again, using Outline view to locate them, I am setting the Employee of the Month Panel container's width property to 33%.

I'm doing the same for the Cafeteria Special and Monthly Events panels.

When I save the file and run the application, you can see that the panels overlap each other.

When I expand the browser, you can see that the panels resize underneath each other. This happens because each Panel container uses absolute

positioning for their upper left corners with explicit x and y values.

Notice that the text in the Employee of the Month panel resizes with its parent container, but the text in the Monthly Events panel does not.

This occurs because the width property of the Label control in the Employee of the Month panel is set to 100%, while the Label control in the Monthly Events panel uses specific values for the width property.

I am returning to Flash Builder.

I am changing the width properties for the Employee of the Month and Cafeteria Special panels back to 250.

I am also changing the width property of the Monthly Events panel's Label control to 100%.

I am saving the file and running the application.

You should see that the panels no longer overlap.

When I resize the browser, the Monthly Events panel expands and the text within the panel resizes to fit the container.

You can mix and match absolute and relative values between siblings in a container.

Once the fixed-sized components are accounted for, the application will divide the remaining space among the percentage-based components.

Also, if you define more than 100% of the available space, all the percentage-based components will be scaled evenly, to maintain the ratio between them.

Back in the main application, I am setting each Panel container's width property to 33% again so that all of the panels have percentage widths of 33%.

I am adding a minWidth property
to the EmployeeOfTheMonth panel with a value of 250 pixels and a maxWidth property with a value of 270 pixels.

Now I’m adding the same properties to the Cafeteria Special panel.

When I save the file and run the application, you can see the first three panels still scale up and down a little as the browser resizes, but they stop scaling when their displays would start to look awkward.

The third component, however, continues to scale to take the remaining area, even though its percentage width exceeds the 33% value that I set.

For your next step, watch the video titled “Using constraints to control component layout”.