Flex in a Week, Flex 4
Video 1.12: Laying out components in containers

In earlier videos, you learned that the Flex MX architecture intrinsically ties the behavior, layout, styles and skins of a component together.

The Spark architecture, on the other hand, explicitly decouples these elements to add more flexibility to your toolkit.

In this video you will learn how Spark lays out components using the layout classes.

You will also learn how these layout classes can be applied to the Spark container classes, to give you fine control over your application display.

There are four Flex framework layout classes: BasicLayout, HorizontalLayout, TileLayout and VerticalLayout.

The BasicLayout class will display all the content in a container based on x and y absolute properties.

The HorizontalLayout class will place all content horizontally next to each other.

The TileLayout class will tile all of the content children of the container.

The VerticalLayout class will layout all of the content children vertically relative to one another.

You will learn more about containers in a bit, but for now, know that each container has a layout property.

The layout property, as shown in this code, is usually added as a nested node in the container.

You can tell that it is a property, not a class, because the word layout is not capitalized.

Inside the layout property tag block, you add an instance of one of the layout classes, like the BasicLayout class, in this case.

Let me be clear that BasicLayout is a class implementation separate from the Application container.

The layout property of the Application container is specific to the container, but the BasicLayout class, is an instance of a layout class applied to this container.

Again, this speaks to the Spark architecture for separating layout from functionality and content.

Here is the code for the starter file in the associated exercise for this video.

It includes a text header display that reads Employee Directory and also four instances of the EmployeeDisplay custom component.

You will learn how to create custom components of your own in a later video, but for now, just treat them as if they are any other Flex component, but in a custom namespace called components.

Note that there are four additional instances that are commented out for later use.

When I run the application, you can see that the components are displayed diagonally from top-left to bottom-right.

When I return to the application code, you can see that the component instances have x and a y properties that explicitly set their absolute position o the screen.

By default, the Application container implements the BasicLayout class to layout its children.

This class uses absolute positioning, which requires that each of the children specifies an x and y property that is based on an origin, or 0,0 point, in the top-right corner of the container.

This illustration shows that the x value increases to the right and the y value increases down the display.

I have located the Properties of parent comment and am creating a layout property for the Application container.

I am typing an opening angle bracket to start my MXML code.

As I start to type the word layout, the code assist tool suggests some options to me.

I am using my mouse to select the layout class and then typing a closing bracket to create the associated closing tag.

Note that the property is in the Spark namespace.

Within the layout tags, I am creating an instance of the BasicLayout class.

Note that I am typing a forward slash and then the closing angle bracket to denote that this tag instance represents both the opening and closing tag.

From an object-oriented programming perspective, you would say that the Application class has a property named layout whose value is an instance of another class, BasicLayout.

You will learn more about OOP in the last video of this Day.

When I save the file and run the application, you can see that the application looks exactly the same since, by default, the Application container already uses the BasicLayout class to lay out its children.

The HorizontalLayout class lays out its children in a single horizontal row.

Since its focus is simply on a horizontal display, it ignores all x and y properties defined on the children.

The height of each row is the height of the tallest child.

Each child calculates its own width by default.
Back in Flash Builder, I am changing the layout class to HorizontalLayout.

When I save the file and run the application, you can see that the x and y properties that are set on each child component are ignored and all content is laid out horizontally.

This includes the Employee Directory title since it is just another child of the Application container like the EmployeeDisplay custom component instances.

Note that the height of a row is the height of the tallest child and that each child calculates its own width.

The VerticalLayout class lays out its children in a single vertical column.

Since its focus is simply on a vertical display, it ignores all x and y properties defined on the children.

The width of each column is the width of the widest child.

Each child calculates its own height by default.
I am returning to Flash Builder and updating the layout class to VerticalLayout.

When I save the file and run the application, you can see that the children are now stacked vertically.

The TileLayout class is a little more complex than the other three layout classes.

It lays out its children in one or more vertical columns or horizontal rows based on a property named orientation.

The valid values for the orientation property are columns, if you want a column layout, and rows, if you want a row layout.

Row is the default orientation.

All cells in a tile layout have the same height and width.

Both values are based on the tallest and widest child, respectively.

I am changing the layout class to TileLayout.

When I save the file and run the application, you can see that the application's children are laid out in a tile configuration, with the Employee Directory text as the first element in the tile.

Because this first child element is so wide, it is forcing the width of the other children to the same size.

Layout classes can be applied to many components, but in this video I will focus on using them with containers.

Containers are essentially rectangular regions of the Flash Player drawing surface.

They are also part of a hierarchical structure of visual objects since a container can hold other components, including controls and other containers.

Already in this video, I have used the term “child” to refer to the content within the Application container.

You saw that the layout classes control a container's children and this includes the child's size and position.

You will learn the basics of component layout in this module, but be sure to watch the videos on Day 4 that continue this discussion of component display and layout.

There are seven available Spark containers that are separated into two categories.

The Group and DataGroup containers are containers without any visual skins applied to them.

If you know that you simply need a container to hold content but does not need a visual skin itself, then you can choose either the Group or DataGroup containers and save the overhead of initializing any visual elements.

The five containers that can have visual skins are SkinnableContainer, SkinnableDataContainer, Panel, Application and BorderContainer.

Of those five, only the Panel container has a default visual skin.

The other four require you to add styles or apply a skin class to them to give them a look-and-feel.

Skinning components is discussed in depth in a later video in the series.

The DataGroup and SkinnableDataContainer are also both discussed in later videos.

Until that point, I will focus on using the Group, Panel, Application and BorderContainer components.

Remember that my application currently looks like this.

It is using a TileLayout layout class, which is unfortunately including the Employee Directory text field.

As I mentioned earlier, the Group container is a very low overhead container that is not skinnable.

As its name implies, its job is to group all of its visual children together.

Remember that the actual layout of the children, is dependent upon the layout class that you assign to the Group container.

I am changing the Application layout class back to VerticalLayout and then adding a Group container around the four custom component instances.

Remember that my methodology likes me to indent any nested elements so I'm highlighting the components and hitting the Tab key to indent them.

I am also un-commenting the additional four custom component instances which I also want to move within the Group container.

I'm going to highlight them all, and then holding down the Alt or Option key, I'm going to use the arrow key to click up so that the content is placed with within the Group container.

You can see that the Group container got moved down here.

Again, I'm going to highlight all these elements and tab them in so I can visually see that they are nested.

I am adding a layout tag property set to the Group container and setting its layout class to TileLayout.

Note that there is now a layout class defined for the application as well as for this new Group container.

A container's layout setting affects only its immediate children.

That means that the Application layout, currently set to VerticalLayout, will only affect the Employee Directory text and the Group container.

They will be vertically stacked on one another.

The TileLayout class instance set in the Group container will only affect the 8 nested EmployeeDisplay custom components.

When I save the file and run the application, you can see that the application is displayed as expected.

The Employee Directory text and the Group container are set vertically stacked and all of the employees are laid out in a tile configuration.

In the earlier video about Spark and MX components, I emphasized that the Spark architecture largely separates the visual styles and skins of a component to a skin class file.

However, if you have some basic styling needs, you can implement it with the BorderContainer class instead of a skin.

This class defines a set of CSS styles that control the appearance of the border and background fill for a container.

The BorderContainer class is a child of the SkinnableContainer class that you place around a Group container.

In Flash Builder I am selecting Help >Flash Builder Help to open the new Adobe Community Help window.

Here in the help files, I am searching for BorderContainer and selecting Local Help.

I am also selecting the ActionScript 3.0 Language and Component Reference.

Now I’m clicking on BorderContainer and then, clicking on styles.

Here in the help files, you can see that this class contains many styles that you can apply to your containers.

Here is the current application.

I want to place a border around all the tiled elements.

Back in the code, I’m adding a BorderContainer instance around the Group container.

As before, I’m highlighting the nested code and hitting the Tab key to indent it.

In the opening BorderContainer tag, I am adding the borderColor property with a value of #0b85b7, which is a shade of blue, a cornerRadius property with a value of 8 and a borderWeight property with a value of 4.

When I save the file and run the application, you can see the EmployeeDisplay components are now surrounded by a rounded rectangle.

For your next step, watch the video titled “Adding scrollbars”.