Flex in a Week, Flex 4.5
Video 5.06: Introducing skinning

In this video, you will learn how to create Spark component skins and work with the skinnable Spark containers.

You will also use constraint-based layout to position the container content within a skin and then apply a skin to a component using CSS.

Here is the completed Employee Portal application for this video.

I will apply three skins for this application: the application gray background, the blue header area and the Panel containers.

The application and header skins are applied inline in the MXML tag instances, while the Panel skin is first applied to all Panel instances using CSS and then applied to only the top three panel instances inline.

This last panel, for the Employee Directory will remained unskinned.

In earlier videos, I emphasized the fact that MX components have their skins built right into the component properties.

Spark skins are different.

Their skins are separate from the content and functionality of the component.

A spark skin can contain graphics, text, images and transitions.

They also support component states so that the component on which the skin is applied can visually respond to user or system events.

You can apply skins to a component in CSS, MXML and ActionScript.

In this video, you will learn how to apply skins using the first two methods.

For MX components, you can apply many styles directly to the component as property values.

Because Spark components are built differently, you only have a few available properties that you can set, like backgroundColor.

If you want to apply styles to a Spark component, you will usually want to create a skin for it.

You will create skins as separate MXML class files and then apply them to a component using the skinClass property.

This example shows that you reference the skin class by first referencing its package.

In this case, the skin resides in the skins directory of the project.

You create a skin class by creating an MXML file by hand or using the File > New MXML Component wizard in Flash Builder.

Usually you will extend the SparkSkin class, rather than the Skin class, in order to take advantage of the Spark look and feel.

You first define all the states of a skin.

By default, you will define a normal and disabled state, but you can create multiple states for your skin.

You will learn how to create a skin for multiple Button states in a later video.

Next, you define a Metadata tag block, which defines the HostComponent directive.

This property declares what type of component this skin will be applied to.

After the HostComponent directive, you specify any graphical elements for the skin.

Lastly, you define the skin parts.

For instance, a container has content and a Panel container has content and a title field.

Doing all this defines the contract between the skin and its component.

In the Flex project, I am opening the skins package to locate the ApplicationContainerSkin.mxml file that I have previously created.

You can see that the opening tag, or in OOP terminology, the class that this skin is extending, is the SparkSkin class.

Otherwise, the code just contains the same namespaces and properties that you've seen in the Application tag.

The first property in this skin is the states property which defines the normal and disabled states.

You must define at least these two states.

Next, you see that I have added a Metadata tag block that defines the HostComponent, which is the Application class in the spark.components package.

You can use the Language Reference in the Flash Builder Help application to find the package for any class.

This HostComponent directive tells the skin which component that the skin can be applied to.

The Rect tag block defines a background for the skin.

You will learn more about MXML graphics in the next video.

For now, just note that there is a fill property with a nested instance of the SolidColor class that sets the color of the rectangle to a shade of gray.

This will create a box around the Application that is 848 pixels wide by 600 pixels high, starting at the 0, 0 point of the application, which is the upper left corner.

Also defined in the Rect tag, are the radiusX and radiusY properties that will round the corners of the rectangle graphic.

The last element in the skin is a required Group container that must have an id property value of contentGroup.

contentGroup is a skin part that represents all of the visual content that you have defined between the opening and closing Application container in the main application file.

You will learn more about skin parts in a later video.

Back in the main application file I am adding a skinClass property to the opening Application tag.

This property points to the ApplicationContainerSkin class, in the skins folder, which I just reviewed.

When I save the file and run the application, you can see that the application has a gray background that butts up right against the left and top edges of the browser.

Back in the skin, I am assigning the horizontalCenter and verticalCenter properties to the Rect tag and setting both values to 0.

This will constrain the application background to the center of the browser.

I am saving the file and running the application.

This is a little hard for you to see in the small recording area, but the gray rectangle is now centered in the browser – both horizontally and vertically – and when I resize the browser, the skin remains centered within the browser; however, the content does not stay centered.

You will be able to explore this better when you do the associated exercise.

Back in the skin file, I'm locating the Group container and adding the horizontalCenter and verticalCenter properties and setting their values to 0.

Like the Rect control, this will constrain the application content to the center of the browser.

When I save the file and run the application, you can see that the application content is almost centered in the brower.

It's slightly askew, where the left side has a larger padding than the right side.

The application looks this way because padding was previously added to the static position values of the content elements.

In the main application file, I am locating the HGroup tag and changing the x and y property values to 0.

To the EmployeeOfTheMonth and EmployeeDirectory custom components – which are the panels in the first column – I am changing the x property to 0.

I am changing the Cafeteria custom component's x value to 275 and the MonthlyEvents custom component's x value to 550.

Finally, I am changing the x property of the Logout Button control to 710.

I am saving the file and running the application.

You can see that both the application skin and the application content are constrained to the center of the browser.

Next I want to make the ApplicationContainerSkin reuseable by designing the edge of the skin's background rectangle to always stay 30 pixels from the edge of the content area..

I am surrounding the Rect control and the contentGroup container with a Group container and indenting the nested tags using the tab key.

I am removing the horizontalCenter and verticalCenter properties from the Rect and contentGroup and adding them to the outer Group container.

This means that the outer Group container will always stay horizontally centered and vertically centered.

From the Rect tag, I am removing the width and height properties And adding the left, right, top and bottom properties with a value of 0.

This will constrain the rectangle to zero pixels to the left, right, top and bottom of the parent Group container.

Now I will add constraints to the contentGroup container so that the container's content will display further from the application border.

I want a nice 30-pixel border of the gray application rectangle around all of the content.

To the opening tag. I am adding the left, right, top and bottom properties and giving them all values of 30.

I am saving the file and running the application.

You can see that the application skin now has a 30-pixel border around the content.

The next task for this video is to create a skin for the background of this header area.

Remember that there are two categories of containers in Spark:

Non-skinnable containers and skinnable containers.

Note that the skinnable containers include the Application and Panel containers plus two other containers named SkinnableContainer and SkinnableDataContainer.

For every Group or DataGroup container that you would like to skin, you should instead use a SkinnableContainer or a SkinnableDataContainer, respectively.

I am returning to the main application file in Flash Builder.

This HGroup container holds the application logo and Employee Portal header text.

I want to apply a skin to this header area with a blue background, so I am converting the HGroup container into a SkinnableContainer and removing the gap property.

I am applying a skin to the container using the skinClass property and setting the value to TopContainerSkin in the skins package.

I am opening the TopContainerSkin class file, which I've already created, by clicking on the class name while holding down the Control key.

In this skin, you see the normal and disabled states as before.

The HostComponent now creates a contract with the SkinnableContainer class.

The Rect block draws a blue rectangle with radiusX and radiusY properties that draw rounded corners.

The last element in this skin is the required Group container with the id property set to the contentGroup skin part.

Remember that this skin part represents all of the content in the container.

In this case, that would be the logo and Employee Portal text.

I'm going back to the main application file to save it and then run it, you see that the main header area now has a blue background with rounded corners.

The last feature I will show you in this video is how to apply a skin in CSS to update the look-and-feel of all the Panel containers at once.

I am opening the PanelContainerSkin.mxml file.

You can see that it has four States and a HostComponent metadata directive that makes a contract with the Panel container.

The Panel skin is drawing a rectangle with rounded corners and a gray background.

I am opening the CSS file for this project, which you will recall, is hooked into the main application file via this Style tag.

In the CSS file, I am locating the Panel container style, which already defines a gray background color.

I'm deleting that backgroundColor attribute and adding a skinClass property with a value that references skins.PanelContainerSkin class.

When I save the file and run the application, you can see that the Panel containers are now skinned with the drawn gray and orange shapes, but the Panel titles are not positioned correctly.

Back in the PanelContainerSkin.mxml file, I am locating the titleDisplay skin part.

I am changing the left property to have a value of 35. This will shift the panel title to the right of the orange box.

To the Label tag, I am adding the color property and giving it a value of #FFFFFF. This will change the panel titles to white.

I am saving the file and running the application again.

You can see that the panel titles are white and they display to the right of the orange box.

Back in the main application file, I have located the Label control with the text Employee and adding the color property and setting the value to #FFFFFF.

To the Label control with the text Portal I am adding the same property and color value.

I'm saving the file and running the application so you can see that the application header is now white.

Lastly, I want to only apply the panel skin to these three containers and not the Employee Directory container.

Back in the application's CSS file, I am commenting out the skinClass property and value in the Panel selector.

I am saving the file and returning to the main application file.

To the all of the custom components, except the EmployeeDirectory component, I am adding the skinClass property with a value of skins.PanelContainerSkin.

I am saving the file and running the application.

You can see that all of the panels are skinned except for the Employee Directory panel. Notice that the background color of the Employee Directory panel is white.

I am returning the main application file.

To the EmployeeDirectory component instance, I am adding a backgroundColor property with a value of #E8E8E8.

When I save the file and run the application, you can see that the Employee Directory panel has a gray background color even though it is not skinned like the other panels.

For your next step, watch the video titled “Drawing with MXML graphics”.