Flex in a Week, Flex 4.5
Video 4.07: Creating pages with Flex states

View states allow developers to create different page layouts in the same application or different layouts within custom components.

In this video, you will learn how to create states and how to build them using Flash Builder's Design mode and Source mode.

You will also learn how to control the display of the states through event handlers.

HTML pages often present a very disjointed user experience because pages need to refresh as new links and content are requested.

This mobile device store provides a very different experience.

Users are smoothly presented with content without a page refresh, and maintain a dialog with the application through interactive elements, like this search interface, that give them immediate feedback and access to more information.

The application is organized into discrete sections, or states.

For instance, when I mouse over a phone I access a new state of the component that displays these additional buttons, and when I click on the phone, I access a new state of the application that gives me details about the phone.

In this video, you will learn how to create view states.

You will learn how to animate between the states in later videos.

Each distinct layout in an application is called a state.

User and system events can be used to change application states.

Between application states, components can be added, modified or removed.

This is the Employee Portal application with the completed code that I will show you in this video.

I have created two distinct states of the application.

The login state displays a form to authenticate users.

When I click on the Submit button, this takes me to main state of the application.

Notice the application is made up of an image control and multiple panel containers.

When I click the Logout button, I return to the login state.

View states can be created and managed visually with Flash Builder’s Design mode or programmatically in MXML code using Source mode.

In this video, I will show you how to create states in both ways, starting with visually, in Design mode.

This is the starter file for the Employee Portal application.

I am switching to Design mode.

Notice the States view and that the default application display contains the main application elements.

If you do not have the States view open, you can access it by selecting Window > States.

I am right-clicking in the States view and selecting New.

In the New State dialog, I am naming the new state loginState and making it a duplicate of the State 1 state.

Now I’m clicking OK and you can see the state listed in the States view.

Changes made to components in Design mode are reflected in your MXML code, so let's talk about these MXML tags.

The states property defines the states in the immediate parent component and wraps all of the defined states in a single block of code.

Each State instance defines the name of one state of the application and takes one property, the name of the state.

The name property is required and must be unique in the given component.

The first state defined in the states property is the default state for the component.

I am switching to Source mode in Flash Builder and locating the states tag block that was created by Design mode.

Here are the two states that I just defined: State1 and loginState.

I am changing the first state's name to portalState.

When I switch back to Design mode, you can see that the first state in the States view is now named portalState.

Currently the content in each state is exactly the same.

I am selecting the loginState in the States view to modify the state.

From the Components view, I am dragging a Panel container and dropping it in the middle of the Design area.

I'm centering it above the other components.

I need to be careful that I am not dragging this new component into one of the other containers.

I want it to appear above the others.

I am switching back to Source mode to verify that it is not nested within another container.

I am giving the new Panel an id property with a value of login and a title property with a value of LOGIN.

I am also changing the value of the width property to 390, the value of the x property to 24, and the value of the y property to 95.

Within the Login Panel container, I am adding a layout property tag block.

Within the layout block, I am adding an instance of the VerticalLayout class to arrange the layout elements in a vertical sequence.

To the VerticalLayout instance, I am adding the gap property set to a value of 10. This will add a 10 pixel, vertical gap between each element.

I'm also adding the paddingTop, paddingBottom, paddingLeft, and paddingRight properties to the VerticalLayout class and setting all of their values to 15.

I am saving the file.

When I return to Design mode, you can see that the Panel container now has a title of LOGIN and the position and width of the container has changed.

From the Component view, I am dragging a Label control into the Login panel.

I am using the Properties view to assign the Text property a value of Username:.

Now I am adding a TextInput control under the Label control and using the guidelines to position it.

I am selecting both them and then copying them and pasting them.

When I paste the controls, the pasted controls are placed directly under the original controls.

I am giving the second Label control a new text value of Password:.

When I switch to Source mode, you can see that the Panel container nests the controls that I just added and it also has an includeIn property that has a value of loginState.

You use the includeIn property of a component to define the state in which the component exists.

If the component exists in more than one state, then you can add each state to the property in a comma-delimited list.

You could alternatively define the excludeFrom property of a component, which declares in which states the component does not exist.

The includeIn and excludeFrom properties of a component are mutually exclusive.

You should not use them together.

Lastly, if you do not define either of these properties, then the component will exist in all states.

With the loginState selected, I am switching back to Design mode and then deleting the Cafeteria Special, Monthly Events and Search Employee Directory panels.

I am selecting the portalState.

You can see that all the components that I just deleted still exist in this state.

When I switch back to Source mode, you can see that the search, cafeteriaSpecial and monthlyEvents Panel instances have the includeIn property set to portalState.

Still in Source mode, I am locating the Logout button and adding an excludeFrom property set to the loginState since the Logout button logically does not belong if you are already logged out of the system.

I am switching back to Design mode and then double-clicking on the Editor tab to maximize the Editor window.

When I switch between the two states using this dropdown list above the Design area, you can see the differences in the display.

So far, I have only switched between the two states in Flash Builder's Design mode.

I will now show you how to do this programmatically.

When switching between states, Flex figures out everything that needs to change between the two states.

You trigger a state change from an event, like a button click or the loading of data.

The key to controlling view states is the currentState property of a component.

As its name implies, this property holds the current view state of the component.

In my example, the component is the Application and its currentState property is the loginState.

To switch between states, you set the currentState property to another view state name.

Within the States view, I am right-clicking on the loginState state and selecting Edit.

Within the Edit State Properties window, I am selecting the Set as start state checkbox and then clicking OK.

You can see the loginState state now shows the word start in parentheses.

When I switch back to Source mode you can see that the opening Application container now has the currentState property set to loginState.

This will ensure that the loginState is displayed by default when the application loads up.

I am locating the Login Panel container and adding a Button control, which I am giving a label value of Submit.

I'm adding the click event to the control and then setting the currentState property to portalState, with single quotes around the name, since double quotes are already being used for the property.

I'm also adding a click event to the Logout button.

The currentState property assignment here is 'loginState'.

When I save the file and run the application, you should see the loginState by default.

When I click on the Submit button, the application switches to the portalState.

When I click on the Logout button, the application switches back to the loginState.

Lastly, back in Design mode, I want to rearrange the two panels in the loginState.

I have selected the Employee of the Month panel and in the Properties view, I am changing the x property to 434, the y property to 95, the width property to 390, and the height property to 200.

This will reposition and resize the Panel container.

I'm also replacing its content with a new message.

To declare different properties for a component in each state, simply reference the name of the state after the property name.

For instance, to create a new text property value for the Label control in the loginState, you would type the property name, text, followed by a period and then followed by the name of the state, which is loginState.

This syntax is also true for component events.

So here is a click event, followed by the portalState.
I am switching back to Source mode and locating the Employee of the Month panel.

You can see that it has the original x and y and width properties, but now it also has x.loginState and width.loginState as well as a height.loginState properties.

Note, however, that the Login panel does not contain any properties that are specific to the loginState.

This is because the Login panel only exists in the loginState, as defined by this includeIn property.

Its properties, therefore, are already specific to the state.

When I save the file and run the application, you can see that both states – and their components – appear as expected.

For your next step, work through the exercise titled "Creating and navigating application states".