Flex in a Week, Flex 4.5
Video 4.10: Animating components with effects

Animation is often considered overused or unnecessary, but, if implemented well, can enhance the user experience and maintain an effective dialog.

In this video you will learn how to use Spark effects to animate components in response to user or system events.

Let me take a minute to define the term “animation” in regards to Flex development.

Animation is the change in a property value of a component over time.

For instance, if you increase the value of the x property of a Panel container, it will move to the right.

If you increase the width and height values of an image over time, the image will grow.

If you decrease the alpha property of a text control, then the text will fade.

The Flex framework provides two packages of effects.

The MX effects are in the mx.effects package and only work with subclasses of the UIComponent class.

Spark effects are in the spark.effects package.

You should choose the Spark effects over the MX effects since they will work on any object, not just those in the UIComponent subclasses.

The Spark effects are based on an Animate super class.

This code shows that you give the Animate instance an id property value to control the animation with ActionScript.

Then you define the id of the object that you want to animate using the target property and binding syntax.

Remember that the definition of an animation is the change in a property over time.

You define the amount of time for the animation using the duration property which, by default, is 1000 milliseconds.

Nested within the Animate instance, you create SimpleMotionPath objects for each property that you want to animate.

For each property, you define starting and ending values or an increment of change, as seen here with the valueBy setting of the width property.

There are many effects in the spark.effects package that are basically wrappers around the Animate class.

They expose the specific properties to animate and create the SimpleMotionPath objects that the Animate class expects.

You can create your own custom effects by extending the Animate class, but the ones in the spark.effects package handle the majority of effects that you will use most frequently.

Spark effects are divided into these 5 categories.

In this video, I will focus on the property and transform effects.

The other categories of effects work similarly but are outside the scope of this training.

I have searched AS Docs for the Spark Fade effect.

You can see that its properties are alphaFrom and alphaTo.

In the lower-left frame, you can see the other effects in the spark.effects package.

You will define the effects in the Declarations block.

Effects are non-visual classes that are applied to visual objects.

You give each effect an id property, which you will use to control when it plays.
Now, let's start coding the Login panel's shaking animation.

Here is the starter file for the application and I have located the Login panel here.

You can see that it has an id property set to login.

I am locating the Declarations block in the Outline view and then instantiating a Move effect that I am assigning an id property value of shake.

The target for this shaking effect is the Login panel, so I am adding the login instance, in a binding to the target property

I want the animation to move just slightly back and forth horizontally so I am assigning the xBy property with a value of 20 pixels.

The xBy property directs the target to move the defined number of pixels horizontally relative to its current position.

Note that if I were to target more than one object with this effect, I would change the target property to targets – with an s – and add array brackets

inside of the curly braces.

Then the ids would be listed with commas between them.

I am undoing this last change since I only have one target object in this case.

To play an effect, you use the play() method for the effect, which is inherited by all subclasses of the Animate class.

In the user or system event handler that will trigger the animation, you reference the effect's id property value, in this case, the shake instance, and then call the play() method.

Of course, if you have more than one line of code for your event handler, you can always create a function and play the animation from there.

Next I am locating the Submit Button control in the login Panel container that I am animating.

I am adding the click event and typing shake.play() for the handler.

I am locating the TextInput control for the password nested within the Login panel and adding the displayAsPassword property and setting the value to true.

This will specify that the text field is a password text field, which means that the input characters are hidden with asterisks.

When I save the file, run the application, and click on the button, you can see that the login panel moves just slightly, exactly 20 pixels, to the right.

Later, I will show you how to make the login screen shake back and forth.

For now, though, I don't want this animation to always play when the user clicks the Login button.

A user should only see the shake animation when the username and password are incorrect.

If the username and password are correct, then the user should be presented with the main portal page.

Back on the Submit Button control, I am removing the shake.play() method call for the event handler and replacing it with a call to the checkLogin() function, which I will create next.

I am scrolling up to find the Script comment and then creating a Script block.

Next, I am creating a private function named checkLogin() that returns a void return type.

You can see here that the Username and Password TextInput controls have id values of username and password, respectively.

Within the function – which I am locating by CTRL + clicking on the function name in the button click handler – I am pressing CTRL+Space twice to show code templates with the content assist tool and selecting the template for the conditional if statement.

In the condition I am checking whether the username.text value entered into the username TextInput control is equal to the string "flex".

I want to only authenticate the user if they enter flex for the username and hero for the password, so I am adding two ampersands for an AND statement in the condition.

Next I am checking whether the password.text property value entered into the password TextInput control is equal to the string "hero".

Here is the states block that you created in an earlier exercise.

Remember that there are two states: loginState and portalState.

The portalState shows the main application display.

If the user properly authenticates, I want to switch the application state to the portalState using the currentState property of the component, which is in this case the main Application container.

Within the condition, I am assigning the currentState property a value of portalState.

Now I am adding an else statement to the conditional statement in order to set the actions for the application when the user does not properly authenticate.

Here I am telling the application to call the shake effect's play() method.
I am saving the file, running the application and clicking on the Login button.

You can see that the animation does move the Login panel 20 pixels to the right.

When I type flex for the username and hero for the password, and then submit the form, the application displays the portalState.

Next, I will show you how to string effects together to create the full back-and-forth shaking effect.

To apply multiple effects at once, you will use composite effects.

The Parallel composite effect causes all effects to run simultaneously while the Sequence composite effect causes effects to run sequentially.

You surround all of your effects with one of the composite effect.

The target of the effect can be defined on the individual effects or on the composite effect.

You can even nest composite effects to generate more complex animations.

You will learn how to do that in the next video.

For now, I am nesting the Move effect within a Sequence effect block.

I'm highlighting the Move effect and pressing tab to indent it.

I am moving the id property and the target property from the Move effect into the Sequence block.

This new composite effect will now contain the entire shake movement.

Next I am copying the Move effect and pasting it seven more times.

To every other Move effect, I am making the xBy property's value negative.

When I save the file, run the application and click on the Login button, you can see that the Login panel now moves back and forth in a shaking motion.

However, this shaking motion moves rather slowly since, by default, an animation takes 1000 milliseconds to play.

Lastly I will show you how to use the duration property of the effect to speed up the rate at which the Login panel shakes.

Doing this will give the animation a more emphatic gesture.

I am adding a duration property to the Sequence effect, and setting its value to 20 milliseconds.

When I save the file, run the application and click on the Login button, you can see that the shake effect animates more vigorously.

Notice that if I continually press the Submit button, the Login Panel container moves to the right.

This occurs because you are executing the shake.play() function while the function is already playing.

I am returning to Flash Builder and locating the checkLogin() function in the Script block.

I am modifying the else statement to check if the animation is playing by adding an if statement to check that shake is not playing.

You may be familiar with the exclamation point as the not operator.

This will only execute the shake.play() method if the animation is not already running.

I am saving the file and running the application.

When I press the Submit button multiple times, the panel shakes, but it no longer moves to the right.

For your next step, work through the exercise titled “Animating components with effects”.