Flex in a Week, Flex 4.5

Video 2.03: Adding event listeners with ActionScript

In the last two videos, you handled events by referencing the event directly in the component's MXML tag.

In this video, you will learn how to handle the events by creating event listeners in ActionScript.

This is the code that defines the event handler in the Employee Portal:Vehicle Request Form that I've been demonstrating in the last two videos.

I am removing the change event from both DateChooser instances so that I can replace them in ActionScript.

There are several important reasons why you would create event listeners in ActionScript.

Some ActionScript classes do not have an MXML equivalent. That means that the MXML syntax for creating the event listener does not exist so you must use ActionScript to register the event listener.

Advanced event handling, which requires adding and removing event listeners, can only be implemented in ActionScript.

Lastly, some developers just prefer the ActionScript syntax over MXML.

When I added a change event to the DateChooser MXML tag for this example, the event was registered with the component when the DateChooser instances were created in the application.

With ActionScript, I can determine the conditions for when I register the event with the component instance.

However, listeners are often added based on some system event, like when a component or the application starts up.

You will learn more about system events in later videos, but I would like to discuss two of them now that are dispatched when the component starts up.

Of the two system events in this discussion, the initialize event fires first, even before the component determines its visual appearance.

Because it fires before the component display is finalized, it is the better choice when your event handler code affects the component's layout or look-and-feel.

The creationComplete event fires after a component has been measured, laid out, and is visible.

Because it fires after a component display is finalized, it is the better choice when your event handler code relies on the component's size, position or appearance.

In my example, I want to register the click event when the DateChooser instances are created and ready for use, so I will add the event listener on the creationComplete event.

I am scrolling up to the top of the main application file to locate the Application container.

In an earlier exercise, you used the creationComplete event to request the data for the application DropDownList control that displays a list of employees.

I am cutting the employeeService.send() method from the creationComplete event and calling a new function named initApp(), which I will create later.

Remember that in my example, the event object is data typed to the CalendarLayoutChangeEvent class.

Also note that the “type” property of the event object is “change.”
This code shows the syntax for adding event listeners to an object.

The first step is to reference the instance name of the object to which the event listener is being added.

You use the addEventListener() method of the EventDispatcher class to register an event handler for a object.

The eventType string references both user and system events. In my example, I am detecting when the user changes the selected date so the event type is “change”.

The second required attribute of the addEventListener() method is the name of the function that should execute when the event is fired.

The event object is created and the addEventListener() method implicitly passes the event object to the event listener function.

Within the Script block, below the dateChangeHandler() function, I am creating a private function named initApp() that takes no parameters and returns a void return type.

Inside the function, I am pasting the employeeService.send() method that I removed from the creationComplete event earlier.

Next, still inside the function, I am referencing the pickupDate instance and using the addEventListener() method to define the handler for the change event.

For the second argument, I am referencing the dateChangeHandler function.

Note that I am not adding parentheses to the dateChangeHandler reference.

This code does not invoke the function, but rather, just references it.

I am copying and pasting this line of code one more time.

And then I am changing the second line to reference the returnDate instance and saving the file.

When I run the application and click on a date in both DateChooser instances, the application works as before.

The dateChangeHandler() function is being called every time the user changes the selected date.

However, using the “change” event type string is not considered a good practice because the compiler will not catch any misspellings in the string, which may lead to unexpected behavior that could be difficult to debug.

The better practice is to use the constant name. If you misspell a constant name in your code, the compiler will catch the mistake immediately. You can easily find information about the constants available in your event by searching for the event in the Language Reference.

I have located the CalendarLayoutChangeEvent documentation and am clicking on the Constants link in the upper right corner to locate the Public Constants section.

You can see that the CalendarLayoutChangeEvent.CHANGE constant is the constant value that you should use for the event type.

Back in Flash Builder, I am updating the addEventListener to use the CHANGE public constant.

I am deleting the change string and pressing the Ctrl+Space keys to force the content assist tool to appear.

When I start typing CalendarLayoutChangeEvent, the public constant appears for me to select.

I am also updating the event type in the second addEventListener() method.

When I save the file and run the application, you can see that the event handlers function normally.

For your next step, work through the exercise titled “Using the addEventListener() method”.