Flex in a Week, Flex 4.5

Video 3.02: Extending the Event class to pass data in the event object

In the last video, you learned how to create a custom event type, in a custom MXML component, and dispatch it on a button click.

You then handled it in the main application to make a second component visible.

In this video, you will learn how to pass data with the dispatched event by extending the flash.events.Event class.

First, you will learn why extending the Event class is necessary to avoid direct binding of data between components.

Once that data is dispatched in the event object of one custom component, you will handle it in the main application and use it in a second component.

In the last exercise, you dispatched an event from the Choose component instance and handled it in the main application to make the Preview component instance visible.

In this video, you will learn how to package the data stored about the selected employee and pass it to the Preview custom component in the dispatched event object.

To accomplish this, you must do more than just create, instantiate and dispatch an event.

To pass data with the event, you must extend the Event class by creating your own ActionScript class to manage the additional data.

You may be thinking, "Why should I go through the trouble of creating an ActionScript class when I can simply bind to the data from one component to the other."

This illustration shows the same Choose and Preview custom components.

If the Choose component has an instance name of chooseEmployee and the DropDownList control within it has an instance name of employeeList, then you can see that it would be easy to simply bind directly to chooseEmployee.employeeList.selectedItem to grab the information about the selected employee for display in the Preview component.

Many developers will fall into this trap, but it's architecturally considered a bad practice.

One of the reasons you create custom components in the first place, is so that you can keep the code and logic separate between them.

If you then start reaching in an out of components to directly access elements inside them, then you have made your code more fragile.

Consider the case that you have two different developers creating these two custom components.

If Joe has written code to directly bind the content of the Preview component to Mark's Choose component, and Mark suddenly decides to rename that component, then Joe now has a broken component.

A much cleaner approach to communicating between components is to loosely couple them.

This is based on the concept that each component is a black box to any other component and its inner workings are hidden.

The only way information is passed into a loosely coupled component is through public properties or arguments.

The only way that information is passed out, is by dispatching a custom event, which is listened for and handled by the application.

Building components in this way enables better re-use and maintenance of the code and is considered a best practice.

To pass data with your event object, you start with the same steps that you learned in the last video.

First you define the event type.

Then you handle a user or system event that triggers the custom event.

In the handler you instantiate the event and then dispatch it to the main application, which handles the event.

In the last video, the application displayed the Employee of the Month Preview component.

In this video, you will attach data to the event that is dispatched to the main application so that the second custom component can use the data.

To create an event object that can carry data, you must create a custom event class by extending the flash.events.Event class.
The reason we need to create a custom class is that the flash.events.Event base class does not support adding properties to the event object.

The event that you created in the last exercise cannot pass custom data and simply notified the application that an event occurred.

In this video, I will extend the Event class to create my own custom event class that will accept the Employee data as a custom property.

You learned how to create ActionScript classes in an earlier video.

To create a custom event class, you will first create a subclass that extends the Event class and then add properties to the subclass.

These properties will become part of the event object.

The class' constructor invokes the super() method of the Event class because you want to access all of the functionality of the Event class and simply add a few custom data objects to it.

Lastly, if you are creating a class that needs to bubble, then you will override the clone() method of the Event class to create a duplicate of the event object for bubbling.

You learned about bubbling in an earlier video, but overriding the clone() method is outside the scope of this series.

Here is the project and main application file for this video.

I will use Flash Builder to help me create the new custom event class, but first I want to create a directory to store the class.

Within the Package Explorer view, I am right-clicking on the src folder and select New > Folder, which is outside your viewing area.

I'm creating a folder named events and clicking Finish.

You can see the folder in my project.

Next, I am right-clicking on the events folder and selecting New > ActionScript Class.

I am naming the new class ShowPreview.

Remember that classes are, by convention named with an initial capital letter.

Don't confuse this class with the showPreview – lowercase s – event type defined in the Metadata tag block of the Choose custom component.

Next, I am clicking the Browse button next to the Superclass field and typing event to select the Event class in the flash.events package.

My new ActionScript class will be in the events package and will be a public class named ShowPreview that extends the flash.events.Event class.

I am leaving the Generate constructor from superclass checkbox selected and then clicking Finish.

The ShowPreview.as file opens in the Editor view.

You can see that the class definition uses the extends keyword to extend the Event class.

It also contains a constructor with the same name as the class and AS file.

The class constructor should do at least four things.

First, it should accept an argument for the event type.

This is required by the parent Event class.

It should also accept arguments for the data you want to pass in.

Next, it should call the super() method to initialize the functionality of the parent Event class for this subclass.

Lastly, it should assign the data passed in via the arguments to the associated class properties.

Here is the ShowPreview ActionScript class again.

Note that the constructor does define the type as the first argument.

Remember that the type argument of the Event class references the type of event that you are dispatching.

For instance, this might be the click type for a Button control.

In the previous video, you created the custom showPreview type for the Choose custom component.

I am not going to use the other two arguments, so I'm just going to delete them.

The constructor also calls the super() method to initialize the functionality of the Event class for this subclass.

Again, I will delete the other two arguments.

The last code I need to add to the constructor are the arguments that will accept the data that I want to pass in the event object.

Let's take a look at the final output again.

You can see that the Employee of the Month Preview component displays the employee's image, name, email and phone number along with a congratulations message.

This information comes from two places in the Choose custom component.

Remember that this DropDownList control is populated from the XML file which contains all the information about each employee in the list.

Although you are displaying only the first name of the employee in the control, all of the employee data is actually stored in the model bound to the DropDownList control.

You can access the data using the selectedItem property of the control.

The congratulatory message comes from this TextArea control here.

I want to create this custom event class to accept both of these values so I am going to create two class properties.

Inside of the class definition but before the constructor, I am typing employeeInfo and pressing CTRL+1 and selecting the Create instance variable 'employeeInfo' option. This creates a private variable named employeeInfo, data typed to the Object class.

I want this variable to be a public variable so I am changing the variable's access modifier from private to public.

This property will contain all of the information about an employee: first name, last name, email, phone number, etc. So I will leave it data typed to the Object class because it will contain many name-value pairs of data.

I am creating a second public variable named message that is a String.

Values for these properties will come from the custom component when it invokes the constructor so I am adding them to the constructor arguments list after the type argument.

Note that I am naming the constructor arguments the same as the class properties.

This is not required and some developers actually don’t approve of this practice.

I, however, do like to name them the same.

I will show you how to distinguish between them in a moment.

Within the constructor but below the super() method, I am populating the employeeInfo class property with the corresponding constructor argument.

However, because both are named the same, I must distinguish between them by adding the this keyword, which, as you learned in an earlier video, is a self-reference for the class instance.

I am also populating the message class property with the corresponding constructor argument.

Now that you have the custom class created that extends the Event class, you can use it in your application.

First you will define this class in the Metadata compiler directive and then you will instantiate it using the new operator.

Next, you will populate the new event object with the custom data that you want to pass and then you will dispatch the event.

The steps here are very similar to what you implemented in the last video, but now you will use the custom class and add some data to the event object.

This is the Choose.mxml custom component file that you modified in the last exercise.

In the Metadata compiler directive, you had created a custom event type named showPreview for the Event class.

I am updating the class reference to use the new custom event class that I just created: events.ShowPreview.

Remember that events refers to the package, which is this events folder and ShowPreview refers to the name of the class.

I am leaving the type name set to showPreview.

It has an initial lowercase s, since this is the event type, not the class name.

In the code you created previously, you handled the click on the Preview button in this function by creating an event object that was dispatched to the main application using the dispatchEvent() method.

This code stays very similar except, now, rather than creating the eventObject variable as an instance of the Event class, I am creating it as an instance of the ShowPreview class.

I am updating the class name in the constructor call as well.

Note that since I used code hinting, Flash Builder did import the class for me.

I have created the event object, but I haven't passed my custom data into it.

The first argument in the ShowPreview class constructor is the event type, which is showPreview, and which I defined in the Metadata directive.

Remember that the second argument in the constructor is the complex Object data type that contains the employee information and the last argument is the congratulations message.

In the Choose custom component, here is the DropDownList control.

You can see that its instance name is employeeList.

The TextArea control for the congratulations message is named message.

In the constructor call, I am referencing these two controls for the two arguments.

First I will add employeeList.selectedItem to grab all of the data stored in the selected item of that DropDownList control which displays the Employee of the Month that was chosen by the administrator.

Next, I am adding the message.text reference to grab the message that the administrator typed into the TextArea control.

I am saving the file to maintain my changes.

Back in the main application, in the last exercise you handled the showPreview event of the Choose component instance with this function.

Currently, all the event handler does is turn on the Preview custom component instance to make it visible.

Let's take a look in the Preview custom component file by holding down the CTRL key and clicking on the link.
You can see that it expects two properties: employeeInfo and message.

The first property is a complex object with multiple name-value pairs of data that contain all the information about an employee.

It is used to populate the image control and other text fields in the custom component.

The second property simply contains the congratulations message for display.

Let's take a moment to evaluate this event object that contains this new custom data.

In the main application file, I am placing a breakpoint on the closing line of the showPreview event handler function by double-clicking in the Editor margin.

Now I am clicking on the Debug button in Flash Builder to run the Debugger.

The application displays in the browser.

I am selecting Athena as the Employee of the Month and writing a quick congratulations message.

Remember that when I click on this Preview button, I am going to start a chain of code logic that dispatches my custom event object and ends with the main application turning the Preview component visible and then hitting my breakpoint.

When that happens, Flash Builder switches me back to the Debugger where I can double-click on the Variables tab to expand it to view my event object.

You can see that this event object is an instance of the ShowPreview event class.

It also contains the two properties employeeInfo and message, which hold the employee information for the selected employee and the typed message from the Choose custom component.

I am stopping the Debugging session and switching back to the Flash perspective.

In the showPreview event handler, I am updating the event argument to be an instance of the ShowPreview class.

You can see that because I deleted the colon and used code hinting, Flash Builder automatically imported the class.

Now I have access to the data that is stored in my event object.

I will populate the previewEmployeeOfTheMonth component with the two properties that it wants: employeeInfo and message.

I am explicitly passing those two properties the two values stored in the associated event object properties.

It is important that you recognize that the main application is handling the transfer of data from the Choose custom component to the Preview custom component via this event handler.

Doing this maintains loose coupling between the custom components, which allows them to be self-contained elements that do not have any hard-coded dependencies.

When I save the file and run the application, you can see the Choose custom component.

I am selecting an employee from the DropDownList control and typing a congratulations message.

When I click the Preview button, the data from this custom component is passed, via the main application, to the Preview custom component, where it is displayed in the UI controls.

For your next step, work through the exercise titled “Extending the Event class to pass data in the event object”.