Flex in a Week, Flex 4.5

Video 3.03: Dispatching a value object from the custom component

At the end of Day 2, you learned how to create ActionScript classes and how to create and use value objects.

At the beginning of this Day of training, you learned how to package your own properties into an event object by extending the Event class.

In this video, you will bring these two topics together by defining a value object as one of the arguments in an extended Event class.

You will then instantiate both the value object and the custom event and dispatch the value object in the event object.

Remember that we have implemented the Employee Portal: Vehicle Request Form using the MVC architecture.

The model of data is stored in this employees class property of the main application and the view, which is the form UI, is defined in this VehicleRequestForm custom component.

The controller controls the business logic of the application and handles the communication between the model and the view.

In this example, the main application, itself, acts as the controller.

It defines the service call for the data, requests the data, processes it, and then passes it to the view as a bound variable to the view's employees property.

In the next set of videos, you will pass the data from this form back to the server.

But right now, that data is stored in the view, as values in this form, and we need to get it back to the controller.

In Flex, the best practice is to implement a loosely coupled component model.

That means that data goes into a custom component in a custom component property, and data comes out of the custom component in a dispatched event.

You have learned all of these concepts in previous videos, but in this video, you will tie all the concepts together in preparation for sending data to the server in the next videos.

As my first step, I will create a value object class to hold all the vehicle request form data.

Here in the form, you can see that this data includes the employee, which I will reference by the employee id property.

It also includes the office and mobile phone numbers, named phone and mobilePhone, respectively, as well as the pickupDate and returnDate instances of the DateChooser controls.

I am going to create the class in the valueObject package, so I am right-clicking on the valueObjects directory and selecting New > ActionScript Class.

Note that the package is automatically filled out for me.

I am naming the class VehicleRequest and leaving the access modifier set to public.

You can leave the other settings alone and click Finish.

The new VehicleRequest.as file opens in the Editor view with this code.

I am going to create the five data properties above the constructor, but in the class definition, as public variables set as String data types.

I am saving the file.

I will create this value object when the user submits the form, so back in the VehicleRequestForm component, I am adding a click event on the Button control and using Flash Builder's code assist tool to generate the handler.

Within the click handler, I am creating a local variable named vehicleRequestData datatyped to the VehicleRequest class I just created.

I need to make sure that Flash Builder imported the VehicleRequest class for me, which it did.

Back in the click handler, I am using the new keyword to instantiate the variable with the VehicleRequest() constructor.

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 last 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, which is named dropDownList with an initial lowercase d.

I am defining the vehicleRequestData.id property with a value of dropDownList.selectedItem.id.

The next two properties are TextInput controls, which I can access through their text properties.

I am setting the value of vehicleRequestData.phone to phone.text and the value of vehicleRequestData.mobilePhone to mobilePhone.text.

The last two controls are DateChooser components named pickupDate and returnDate.

To access their values I must reference the selectedDate property of the control.

I am setting the value of vehicleRequestData. pickupDate to pickupDate.selectedDate.

Note, however, that this value is a Date data type that I must convert to a string using the dot toDateString() method.

I am copying this code and pasting it again and then changing the property references to returnDate.

I am putting a breakpoint on the closing brace of the click handler and then debugging the application.

I want to verify that the value object was created.

I am filling out the form and then submitting it.

This triggers the click handler, which creates the value object and then hits my breakpoint.

In the Debugging perspective, I am maximizing the Variables view and then verifying that the value object was, indeed, created.

Now I'm ready to dispatch this data back to the controller, which is my main application.

I am stopping the Debugger and returning to the Flash perspective.

Remember that a generic event will not be able to pass this data in the event object

I must create a custom class which extends the Event class.

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.

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

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

I am naming the new class VehicleRequestEvent.

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

Next, I am clicking the Browse button next to the Superclass field and typing…oh there it is already….flash.events.

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

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

The VehicleRequestEvent.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 of the class and AS file.

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.

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 is the argument that will accept the data that I want to pass in the event object.

I want to create this custom event class to accept the vehicleRequestData value object.

Above the constructor, I am typing vehicleRequestData and pressing CTRL+1 to invoke the quick assist tool and selecting the Create instance variable 'vehicleRequestData' option. This creates a private variable named vehicleRequestData – which I am arbitrarily naming the same as the value object in the custom component – and data types it to the Object class.

I am changing this variables access modifier to public and using the content assist tool, CTRL+Space, to change the data type to the VehicleRequest class.

Note that Flash Builder has imported the class for me.

This property will be passed to the class in the constructor so I am adding it to the constructor argument list behind the type argument.

Below the super() method, I am typing this.vehicleRequestData equals vehicleRequestData.

Remember that the this keyword denotes that this is the class property while the lack of the this property, represents the 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.

I am saving the Event subclass and then returning to the VehicleRequestForm.mxml file.

In the Metadata comments section, I am adding the Metadata tag and then defining the Event type.

I am naming the event type vehicleRequestEvent – notice the lowercase v – and then setting the type to the event subclass that I just created, which is events.VehicleRequestEvent.

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

Back in the button handler, after the code that populates the value object, I am creating a local variable named eventObject and data typing it to the VehicleRequestEvent class.

I am using the new keyword and calling the VehicleRequestEvent() constructor.

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

In the constructor, I'm passing "vehicleRequestEvent" as the type argument and the vehicleRequestData value object as the second argument.

Next I am using the dispatchEvent() method to dispatch the eventObject instance, which now contains the vehicleRequestData value object.

I am saving the custom component file and then returning to the main application file.

In the VehicleRequestForm component instance, I am handling the vehicleRequestEvent event by using the code assist tool to generate the event handler.

In the handler, I am adding an Alert message using show() method that reads "Your vehicle request has been submitted."

I am saving the file and running the application to verify that the event is being handled and that the Alert message displays.

In the VehicleRequestForm custom component, I'm removing the breakpoint from the submit button handler.

Back in the main application, I am placing a breakpoint on the closing brace of the event handler for the VehicleRequestForm custom event.

I'm then debugging the application to verify that the value object is available in that custom component's dispatched event object.

I am filling out the form and then submitting it to dispatch the event and run that event handler.

In the Debugging perspective, you can see that the VehicleRequestData value object was, indeed, created in the event object.

In the next videos, you will learn how to use the RPC components to send this data to the server.

For your next step, work through the exercise titled "Dispatching a value object from the custom component".