Flex in a Week, Flex 4.5
Video 4.05: Creating item renderers and item editors

Earlier in this Day of training, you created an item renderer for the DataGroup container and then applied it to two Spark List controls.

In the last exercise you used the DataGrid control to display data.

In this video, you bring your knowledge of item renderers to the DataGrid control and learn how to create inline item renderers and item editors.

You will also learn how to create component item renderers and editors.

Then you will learn how to set an item renderer as an item editor.

Lastly, you will learn how to handle a user click on a cell in the DataGrid control.

This is the application that you built in the last exercise.

By default a DataGrid control displays only one data field per column, but you implemented a formatter to display two data fields in this first column.

Note, however, that you displayed these two data fields in one text string by concatenating them together.

In order to display components in a DataGrid column, you must create an item renderer for the column.

In this video, I will create this item renderer for the first column, which displays not only the employee name data, but also an image of the employee.

I will also show you how to handle a click on a record in the control.

Lastly, I will create an item editor, which allows users to edit the data in the DataGrid control.

This is the main starter application file for this example.

You can see that I am retrieving employee information in this HTTPService call and then populating the employeeList ArrayCollection variable with the data using the result handler.

When I run the file, you can see that the DataGrid displays the employee's full name and hire date.

Back in the code, I am adding a third instance of the GridColumn control and setting the dataField property to evaluation and the headerText

property to Evaluation.

Remember that the dataField property references the data from the employees.xml file in the data package of this project.

This data represents the quarterly performance evaluation score for each employee.

When I save the file and run the application, you can see that a third column was added, but the column is very small.

I'm returning to Flash Builder, and to the DataItem instance in the typicalItem property, I am referencing the evaluation data object and setting its value to Evaluation.

This will make the last column as wide as the word Evaluation.

When I save the file and run the application, you can see that this is true.

Now let's take a closer look at item renderers and item editors.

The DataGrid control uses one text field as the default item renderer and one text input field as the default item editor.

You can create a custom item renderer or editor to customize the appearance and behavior of your DataGrid control.

Visually, item renderers and item editors behave differently.

An item renderer is used as the default display, while an item editor is only used when an item is being edited.

You can set the item renderer to act as an item editor by setting the rendererIsEditor property to true.

The data that is stored in the dataProvider property of the DataGrid control for each record is available to an item renderer or editor in the data property.

Since renderers and editors can display multiple data values, you control the value that is being edited by the item editor with the value property and make sure that you make it a two-way binding.

This will ensure that the edited value is updated in both the data provider and the item renderer display.

You can implement item renderer and item editors in two ways:

As inline or component code.

Note that the only difference between the two is structural – the former puts code inline to the DataGrid instance while the latter puts the code into a component file.

The first step to create an inline item renderer or editor is to create the itemRenderer or itemEditor properties as nested properties of the GridColumn instance, rather than as properties in the tag.

Within the nested property tag block, you will then add a Component tag block.

Within that, you can add your visual controls, which can be simple or complex displays.

This code sample shows an itemEditor property nested inside a GridColumn block.

Within that is a Component block, that specifies a new scope for the item editor.

Within the Component block, is the GridItemEditor class instance.

Within this class instance, you can place a control for display.

This is very similar to creating item renderers for the DataGroup container that you learned how to create at the beginning of this day of training.

If you have more than one control to display in your editor or renderer, you should place the controls within a container.

This code, now for an item renderer, shows the Label and BitmapImage controls nested within a VGroup container.

It also shows the use of the itemRenderer property, instead of the itemEditor property and the GridItemRenderer class, instead of the GridItemEditor class.

Back in the main application file, I am changing the Evaluation GridColumn instance from single tag syntax to block syntax.

Within the GridColumn block, I am adding an inline itemRenderer property block.

Inside the itemRenderer block, I am adding a Component block and within that I'm adding a GridItemRenderer block.

In the GridItemRenderer block, I am adding a Label control and giving it a text property that binds the value of the data object’s evaluation.

Remember that all data stored in the dataProvider property is referenced through the data property when you’re inside of an item renderer or item editor.

Next, I am adding the height property with a value of 100%, the right property with a value of 10, the verticalAlign property with a value of

middle, the fontWeight property with a value of bold and the fontStyle property with a value of italic.

I'm saving the file and running the application.

You can see that the text in the Evaluation column of the DataGrid control still displays the evaluation data and has been formatted.

A complex inline item renderer might be better built as a separate component.

If you create a component item renderer, you will have greater flexibility and functionality.

In addition, it is easier to reuse your item renderer if it is a component.

You have to build your item renderer or item editor component in a separate file.

These components can be written in either MXML or ActionScript.

The data for each item in the column is referenced as the data property and is passed to the component whereby the component determines how the data is displayed.

Back in Flash Builder, I am right-clicking the src folder and selecting New > Item Renderer.

In the New Item Renderer dialog box, I am typing components for the Package and EmployeeDisplay for the Name.

I’m selecting Item renderer for Spark DataGrid from the Template dropdown list and clicking Finish.

The EmployeeDisplay component has opened in the Editor view.

I’m deleting everything between the GridItemRenderer tags and adding an instance of the VGroup container.

Within the container, I'm adding a Label control and giving it a text property that uses the firstName in the data object, followed by a literal space and then the data.lastName.

Under the Label control, I am adding a BitmapImage control and setting the source property value to images/{data.id}.jpg.

To the opening GridItemRenderer tag, I'm adding width and height properties with values set to 100.

Now, I am saving the file and switching to the main application file.

I am removing the labelFunction property from the GridColumn instance that displays the employee's name.

To the same instance of the GridColumn, I am adding the itemRenderer property and setting its value to point to the EmployeeDisplay custom component in the components package.

When I save the file and run the application, you can see the Name column of the DataGrid control now displays both the employee names and images.

Notice that the names and images are not centered and you may not be able to see the whole DataGrid control in the browser.

To the DataGrid control, I am adding a height property with a value of 600.

I am saving the file and switching to the EmployeeDisplay component file.

To the opening tag of the VGroup container, I am adding the horizontalAlign and verticalAlign properties and setting their values to center and middle, respectively.

Next, I am adding the width and height properties and setting both values to 100%.

I am saving the file and running the application.

You can see that the employee data in the Name column is now centered within the column and the DataGrid control is contained within the browser.

Notice that the DataGrid control is not editable.

As you know, the display that you view in a cell of the DataGrid control is an item renderer.

When you double-click on an editable cell, you are presented with the display for an item editor.

Like the item renderer, by default, the item editor is a text field.

Next, I will apply the NumericStepper control as an item renderer for the Evaluation score and then set it to display as the item editor as well.

So, back here in this code, I am adding the editable property to the DataGrid instance with a value of true.

I am saving the file and running the application.

Notice that every cell in the DataGrid is editable if I double-click on it.

I am returning to the main application file.

In a DataGrid control, to make any column editable, you must turn on editability for the entire control and then turn off the editability on the columns that should not be editable.

I am adding the editable property to the employeeName and hireDate fields and setting the value to false, since I only want the Evaluation column to be editable.

When I save the file and run the application, you can see that only the Evaluation column is editable.

Back in the main application file, I am locating the GridItemRenderer block for the Evaluation column and commenting out the Label control within the block using the keyboard shortcut CTRL + SHIFT + C or CMD + SHIFT + C on the Mac.

Within the GridItemRenderer block, I am adding an instance of the NumericStepper control, saving the file and running the application.

The Evaluation column now displays a NumericStepper control for the item renderer, but all of the values are set to zero,

I am returning to the main application file, and adding the value property to the NumericStepper control and giving it a value that binds the data.evaluation data.

When I save the file and run the application, you can see that the Evaluation column now shows the correct data.

Also notice that I can change the value in the NumericStepper, but if I change the value to a number larger than 10, it switches back to 10.

This occurs because the NumericStepper has a default maximum value of 10.

I'm returning to the main application file.

To the NumericStepper control, I am adding the maximum properties and setting its value to 100.

I’m also setting the width property to 100% so that the control takes up the entire column width.

When I save the file and run the application, you can see that the component now fills the column.

You will also note that the maximum number in the NumericStepper is now 100.

If I type a number greater than 100, the NumericStepper will switch the number to 100 as it did before.

When I double-click the NumericStepper, notice that the item editor is still a text field.

To the GridColumn control for the evaluation data, I am adding the rendererIsEditor property and setting its value to true.

I am saving the file and running the application.

When I double-click an item in the Evaluation column, you can see that there is no longer a text field.

This occurs because the item editor is now the NumericStepper control defined in the item renderer.

Now, I want to use the itemEditor property to create an inline item editor.

Back in the main application file, I am removing the rendererIsEditable property from the evaluation GridColumn instance.

I am locating the inline itemRenderer property and changing it to the itemEditor property.

I am also changing the GridItemRenderer instance to an instance of the GridItemEditor.

I am saving the file.

You can see that I get a warning that says, "Data binding will not be able to detect assignments to 'data'."

I am running the application.

Notice when I try to edit the last column, the value in the NumericStepper is zero.

Also notice that when I change the number it returns to the original evaluation score.

I am returning to the main application.

To the NumericStepper control, I am changing the value property to @{value}.

The "value" in the binding is a property of the GridItemEditor instance and holds the value of the data that is being edited.

You use @ to create a two-way binding so that when a user updates the NumericStepper, the data in the data provider updates, too.

When I save the file, the warning goes away.

I am running the application.

Now when I edit the Evaluation column, the value in the NumericStepper starts at the original evaluation score.

When I change the value, the NumericStepper is properly updated.

Now, I want to create a component item editor.

Back in Flash Builder, I am right-clicking the components package and selecting New > MXML Component.

In the New MXML Component dialog box, I am typing EmployeeEvaluation for the Name.

I am basing the component on the GridItemEditor class and removing the width and height values.

I am clicking Finish.

The new component file opens in the Editor view.

I am returning to the main application file.

I am cutting the NumericStepper control and pasting it under the Declarations block in the EmployeeEvaluation component file.

I am saving the file and returning to the main application file.

I am removing the itemEditor property block and its contents and modifying the Evaluation GridColumn instance so that it uses single tag syntax.

To the same GridColumn instance, I am adding the itemEditor property and setting its value to reference the components package and the EmployeeEvaluation class.

When I save the file and run the application, you can see that the application has not changed.

For the last section of this video, I will show you how to capture and handle a user click on a cell.

When a selection is made in a DataGrid control, a selectionChange event is broadcasted.

The item selected is in the selectedItem property, which holds the data for the selected record in the dataProvider property.

Now I will show you how to handle the selectionChange event when a record in this DataGrid control is clicked.

I will also show you how to display an alert message that tells you the name of the selected Employee when you click the record.

To the DataGrid control, I am adding the selectionChange event and generating the event handler, which I am CTRL + clicking on.

Within the selectionChangeHandler() function, I am removing the generated code stub, and using the content assist tool (CTRL+Space), to add the Alert.show() method.

You can see that because I used content assist to add the Alert.show() method, the Alert control was automatically imported.

Within the Alert.show() method, I am adding the string "You selected" concatenated with the value event.currentTarget.selectedItem.firstName.

Separated by a comma, within the Alert.show() method, I am adding a title heading that reads Employee Portal: Notification.

I am saving the file and running the application.

When I double-click a cell in the Evaluation column, you can see an alert message appears, displaying the name of the employee I selected.

For your next step, work through the exercise titled “Creating and using item renderers and item editors”.