Flex in a Week, Flex 4.5
Video 4.04: Creating and formatting the DataGrid control

The DataGrid control is a Spark component which displays data in columns and rows like a spreadsheet.

In this video, you will learn how to bind data to the DataGrid control and then define which data fields to display in columns.

Next you will combine two data fields into one column display and, lastly, you will format the column data using a Formatter component.

This is the DataGrid control that I will implement in this video.

The data provider that is bound to the control has much more data in it, but I will only display three data properties in the two columns that you see here.

The first column concatenates the first and last name fields together and the last column displays the hire date field based on a DateTimeFormatter definition.

The Spark DataGrid component is a skin around the Spark Grid control.

It can show multiple columns of data and can include columns that are editable.

It displays column headings above a scrollable grid of cells organized in rows and columns.

Some of the DataGrid control features include:

Interactive column width resizing
Control of column visibility
Cell and row selection
Single and multiple item selection modes
Customizable column headers
Cell editing
Column sorting
Custom item renderers and editors
Custom DataGrid skins
Smooth scrolling through large amounts of data

You will learn how to use many of these features in this video and the next.

This is a code sample for how to implement the control in MXML.

Data is provided to the DataGrid control as a complex collection of objects with multiple name/value pairs.

You use the dataProvider property to pass this collection of objects to the DataGrid control.

The dataProvider contains the data model for the control and consists of a list of objects called items.

Each item is displayed as a row in the DataGrid control.

Remember from previous discussions, that the best practice is to bind the dataProvider property to an instance of the ArrayCollection or ArrayList class.

Both classes are watched in binding statements, but the ArrayList class is lighter in weight and is typically used when you are not implementing any advanced sorting of data.

On the other hand, you must use the ArrayCollection class for column sorting.
Each property in the ArrayList or ArrayCollection instance is displayed in a column of the DataGrid control.

By default, the columns are displayed in alphabetical order and the column headers are set to the element names in the dataset that you defined in the dataProvider property.

This is the starter application file for this video.

You can see that this HTTPService call retrieves employee information from a local XML file in the data directory of the project.

The result handler for the HTTPService call uses inline ActionScript and the lastResult property to populate a class property named employeeList that is an ArrayCollection instance.

Below this Label control, I am creating a Spark DataGrid instance with a dataProvider property that binds to the employeeList ArrayCollection instance.

When I save the file and run the application, you can see that all the data from the XML file is displayed.

The header text for each column is the name of the associated node from the XML file.

Note that by default, I can click on a column header to sort the data.

Now, I will show you how to add alternating colors to the rows of the DataGrid control.

Back in the main application file, I am adding the alternatingRowColors property to the opening tag of the DataGrid.

I am setting the value to #FFFFFF and #CCCCCC, separated by a comma within Array brackets.

Note that you can add more than 2 colors, but I want to keep this application simple.

I am saving the file and running the application.

You can see that the rows of the DataGrid now display alternating row colors of white and gray.

If you want to modify the default display of the DataGrid control, then you must create an instance of the GridColumn class for each column of the DataGrid control.

When you explicitly create each column, you can limit the number of columns to display, set the order in which they are displayed, change the header text and control the formatting of the column content.

This is a code sample for how to implement the column in MXML.

To use the GridColumn class, you first create a columns property for the DataGrid control.

Within the columns property block, add an ArrayList or an ArrayCollection instance.

Remember that you should use the ArrayList class if you are not implementing any advanced sorting of data.

Then, within the ArrayList or ArrayCollection class, you create one or more GridColumn instances that reference the dataField from the dataProvider to display along with the headerText property for the column header.

I am turning this control into block syntax and then creating a columns property inside the DataGrid tag block.

Remember that the dataProvider property contains the data model for this component.

The column displays are the view.

In other words, all of the data is stored in the dataProvider property and you are merely defining which pieces are exposed via the columns.

Within the columns block, I am adding an ArrayList block.

Inside the ArrayList block, I am adding a GridColumn instance with a dataField property value that points to the lastName field of the data provider.

I am also adding the headerText property with a value of Name.

I am creating a second GridColumn instance with a dataField property value of hireDate and a headerText property set to a value of Hire Date.

When I save the file and run the application, you can see that there are only two columns of information displayed in the DataGrid control.

Notice that some of the content within the Name column has been cutoff.

I will show you how to fix this later in the video.

I would prefer that the first column display both the first and last name.

To do this, I need to use the labelFunction property to reference a user-defined function.

The labelFunction is a property of the GridColumn class that specifies a user-defined function to format column data.

Note that you do not use the parentheses in this function name since it is a reference to the function, not an invocation of it.

The labelFunction property is called once for each row in the DataGrid control and can use all the data for the entire item, not only the data in a column's dataField property.

When you create a formatter function, you must specify two arguments for it.

The first argument is an Object instance that is traditionally named item and represents all of the data in that record of the dataProvider object.

The second parameter is traditionally named column and is data typed as an instance of the GridColumn class.

It represents the name of the column on which this function is being called.

You will use the return statement to send back a formatted string.

At the end of the Script block, I am creating a private function named **employeeName** that returns a String variable.

The function takes two arguments.

I am naming the first argument item and data typing it to the Object class.

This function will be called for every row in the DataGrid control.

Each time it is run, the DataGrid control passes all of the data in that one row to the function.

This item argument represents all the data in the one row that is currently being evaluated by the function.

That means that within this function, you can use the item object to reference multiple data fields for display in the column.

I am naming the second parameter for this function column and data typing it as a GridColumn instance.

This column argument references the name of the GridColumn instance that calls this function.

I will show you how to use it effectively in a few minutes.

I am typing return and adding the item.firstName value, concatenating a literal space and then concatenating the item.lastName value.

Back in the DataGrid control, I am deleting the dataField property for the first GridColumn instance since this references one specific field.

I’m replacing it with labelFunction property and referencing the employeeName function.

When I save the file and run the application, you can see that the Name column now has the first and the last names for the employees displayed, but the width of the column isn’t large enough.

You can set the column widths in a DataGrid control by adding the width property, but an alternative to this method, and a better practice, is using the typicalItem property.

The typicalItem property allows you to specify example data to set the widths of a DataGrid control's columns.

You must add an instance of the DataItem class to the typicalItem property to define your example data.

The DataItem properties must match the values you specify for the dataField properties in the DataGrid.

You should provide a value for each property that is representative of the data because it will determine the width of the column.

If there are two data fields used in one column, then the column width accounts for both their typicalItem definitions.

This code shows an example of how to implement the typicalItem property within a DataGrid control using a DataItem instance.

Above the columns block in the DataGrid control, I am adding a typicalItem property block.

Within the typicalItem block, I am adding an instance of the DataItem class.

To the DataItem instance, I am referencing the firstName data object and setting its value to Christopher because it is a string that is longer than any of the data values.

Next, I am referencing the lastName data object and setting its value to Winchester.

Finally, I am referencing the hireDate data object, and setting its value to 12/31/2011.

When I save the file and run the application again, you can see that the column fits all the data.

The first column is now as wide as it must be to fit the name Christopher Winchester.

Next, I want to improve the display of the hire date field by applying a Flex formatter to the data.

To do this, I need to use the labelFunction property again to reference a user-defined function.

In this case, I'm naming it dateFormat.

Remember that formatters may affect the visual display of an element but they are not visual elements themselves, so you should create them in the Declarations block.

You learned about formatters in Day 3 of training.

I am creating a DateTimeFormatter instance below the HTTPService object.

I am giving the formatter an id property of employeeDateFormatter.

I am also applying a mask with the dateTimePattern property that has a value of d-MMM-yy.

This will always display the day as a one- or two-digit value, the month as a three character abbreviation, and year as a two-digit value.

After the employeeName() function in the Script block, I am creating a private function named dateFormat that returns a String variable.

The function takes two arguments – the item and column, like this one – so I'm just going to copy this one and paste it.

I am typing return and then referencing the employeeDateFormatter instance.

I am using its format function and passing it a value of item.hireDate to format the hire date.

When I save the file and run the application, you can see that the hire date column now displays the formatted date.

The function I created worked just fine, but only for the hireDate column.

In many cases, you will want to have a generic function to handle the formatting for multiple data columns.

So instead of using a reference to a specific column name, you can reference the dataField property of the GridColumn class.

By doing this, you will avoid having to write multiple column-specific functions.

The dataField property references the name of the dataField property associated with the column you are referencing.

Note that in this code I have hard-coded the hireDate field in the format() statement.

Remember that the column argument represents the name of the column on which this function is being called.

If you look at the GridColumn instance that is calling this function, you can see that it is displaying the hireDate data field.

This means that the column object is the hireDate.

I am updating the hireDate reference in the format() function to reference square brackets, which will allow me to evaluate a dynamic variable.

In this case, I'm evaluating the column.dataField property.

This syntax translates into the statement:

Look in the row of data and find the column that is calling this formatter, and then apply this DateTimeFormatter instance to it.

This is slightly more code, but it is also more flexible code.

This change will allow you to use this same dateFormat() function for other columns of the DataGrid because the column.dataField will now always refer to the proper column.

When I save the file and run the application, you can see that there is no change to the application display.

For your next step, work through the exercise titled "Using the Spark DataGrid control".