Flex in a Week, Flex 4.5
Video 3.05: Using formatters

In the last three days of training, you learned how to retrieve, send and handle data.

In this video, you will learn about the formatter classes that you can use to format this data for display.

Here is the Employee Portal: Vehicle Request Form application before I apply any formatters.

When I click on one of the DateChooser components, the date in the Alert dialog is formatted with the day and month abbreviated strings.

I want to change the format so that it is displayed all in numbers, like this.

I will also show you how to apply a phone formatter.

Before I apply the formatter, I can type letters into the phone field, and when I type in the ten digits for a US phone number, nothing happens to fix the display.

After I apply the formatter, it will no longer accept text entry.

When I clicked out of the text field to trigger the formatter, the text string is deleted.

When I type in the ten digits to create a US phone number again, and then click away, the number is formatted appropriately.

The formatters are components used to format data into strings.

The Formatter class declares a format() method that takes a value and returns a string.

You will generally want to trigger a formatter just before you display the data.

There are both MX and Spark formatters that are implemented similarly.

Here is a list of the available Spark and MX formatters.

You should use the Spark formatters whenever possible.
Spark formatters are designed to take advantage of locale-specific formatting of dates, times, numbers and currencies.

This enhances your ability to create globalization-ready applications and is made possible through the globalization APIs available in Flash Player 10.1 and referenced through the flash.globalization package.

Spark formatters give you access to the locale settings in the operating system on the actual device, while MX formatters are limited to locales provided by the Flex SDK or created by the developer.

The details of locale implementation are outside the scope of this course.

While the formatter classes do affect the visual display of data, they are not actually visual objects themselves.

Therefore, in order to create an instance of a formatter class, you must place it in the Declarations code block where all non-visual elements are added.

When you are ready to use the formatter, you will call its format() method.

All formatter subclasses have a format() method.

Each formatter subclass has specific properties to customize the string presentation.

Here is the Spark CurrencyFormatter class, which formats a valid number as a currency value.

You can adjust the currency symbol, decimal formatting, locale and more.

You can also place a currency symbol, which is a dollar sign by default, on the left or right side of the number using the currencySymbol property.

This CurrencyFormatter instance displays a dollar sign but aligns it to the right of the dollar amount.
The Spark DateTimeFormatter class has fewer properties.

The dateTimePattern property defines a pattern string, or mask, for how the date will be formatted.

I will discuss that further in a minute.

This is the starter file for the example.

Remember that all the UI elements in the application are created in the VehicleRequestForm custom component, which is the view in our MVC implementation.

The form includes some form components, including these two DateChooser controls.

They are named pickupDate and returnDate, respectively.

At the beginning of Day 2, you registered a change event on both of the controls that calls this dateChangeHandler() function when the user selects a date from the calendar.

The date is referenced and displayed through the event object here: event.target.selectedDate.

Next, I will create a date formatter and apply it to that display.

So, inside the Declarations block and below the HTTPService object, I am creating a DateTimeFormatter instance with an id property value of requestDateFormatter.

All formatters contain pattern strings specific to the data displayed.

This help information shows the DateTimeFormatter class in the flash.globalization package.

Specifically, I am looking in the public methods for the setDateTimePattern() method.

When I click on it, you can see all the acceptable pattern pattern strings for the DateFormatter class.

You create a pattern for the date display by using the specific letters listed here.

For instance, yyyy will format the year as a four-digit number.

Capital M will produce the month as a number without a preceding zero.

Lowercase dd will produce the day of the month with a preceding zero.

You specify the pattern string in the dateTimePattern property of the DateTimeFormatter instance.
I am adding the dateTimePattern property to the DateTimeFormatter instance with a value of MM-dd-yyyy.

To apply the formatter to the date, I am locating the dateChangeHandler() function again and wrapping the event.target.selectedDate string value with the requestDateFormatter instance's format() method.

I am saving the file and running the application.

You can see that the date is now formatted as expected when I click on a date in the calendar.

Now I will add a formatter for this Mobile Phone number field.

Here is the help information for formatting phone numbers with the PhoneFormatter class.

You can see that you use the # sign as a designator for the numbers and then add other characters, like the parentheses or the dash, to the format pattern.

I will implement this formatter, with the parentheses.

I will also add a dash right here.

In the Declarations block, I am adding a PhoneFormatter instance with an id property value of phoneFieldFormatter and a formatString set to (###) ###-####.

I want to trigger this formatter after the user has typed the number in the mobile phone field and then has clicked outside of it, or removed focus from it.

I am adding a focusOut event to the mobilePhone TextInput control and I am using Flash Builder to generate the handler.

In the handler, I am going to replace the mobilePhone.text property value with the phoneFieldFormatter.format() string.

Specifically, I will format the mobilePhone.text value and then apply it back to itself.

I am saving the file and running the application.

When I type in the ten digits for the US phone number and then click outside the text field, you can see that the formatter is properly applied.

If I delete the phone number from the Mobile Phone field, type 555 and remove focus from the field, you can see that the text I entered is deleted from the field.

This happens because the formatter returns a blank field when the input is invalid.

The only valid string is a 10-digit number without any characters.

Now, I will show you how to add a custom validation check to return an Alert message if the input to the formatter is invalid.

Back in the mobilePhone_focusOutHandler() function, below the mobilePhone formatter, I am typing if and pressing CTRL+Space twice to show a list of code templates with content assist.

I am selecting the template for the if statement.

I will check to see if the text entered in the Mobile Phone field is invalid by adding phoneFieldFormatter.error as the condition of the statement.

Within the conditional statement, I am adding the Alert.show() method to display the text "Phone format error:" concatenated with the phoneFieldFormatter.error.

I am saving the file and running the application.

When I type 555 in the Mobile Phone field and remove focus from the field, the formatter still returns a blank field because the number I entered was invalid, but an alert message also displays indicating that I entered an invalid string.

You can use what you've learned here, to apply a formatter to the office phone field.

For your next step, work through the exercise titled "Using formatters".