Flex in a Week, Flex 4.5
Video 3.06: Validating form data

As an experienced web developer, you have probably implemented both client-side validation and server-side validation.

Earlier in this Day of training, you saw validation messages being sent from the server for you to handle in your Flex application.

In this video, you will learn how to perform client-side validation in your Flex application.

However, first you will learn how to add required indicators and help content to the form.

Then you will use the Validator subclasses, which you will instantiate and trigger with event handlers.

Lastly, you will use the validate() and validateAll() methods of the Validator class to trigger validation with ActionScript commands.
This is the Employee Portal: Vehicle Request Form application.

Note that the submit button for the form is outside your field of view.

You will just have to trust me when I say I am clicking the submit button.

The asterisk indicates that the field is required, although, you will find that this is just a display implementation.

Applying the asterisk alone, doesn't actually perform any validation.

The text to the right, here, represents help content.

I have added validators to both these phone number fields to ensure that the user has entered a valid number.

When I submit the form without typing any data, red outlines appear around each of the form fields, indicating that an error has occurred.

To the right of the fields, you see the validation messages stating that the fields are required.

If I enter invalid data into any of the fields, like a letter into the Office Phone field, and then click the submit button again, you will see that the appropriate error message is displayed.

This is the starter file for the application.

Remember that the form display is actually in the VehicleRequestForm custom component, which acts at the view in our MVC implementation.

I am Control + clicking on the component name to open the associated class file.

In the form UI code, I am adding the required property to the Mobile Phone FormItem container and setting its value to true.

This code will only add a required asterisk to the display and does not perform any validation.

I am saving the file and running the application.

You can see that when I click in and out of the Mobile Phone text control, or hit the Submit button, which is below your field of vision, nothing happens.

This demonstrates that the required property of the Mobile Phone FormItem control only adds the required asterisk to the display and does not perform any validation.

You can also display help content, which provides a description or instructions for each FormItem container instance.

The help content will be displayed to the right of the FormItem container content but is coded inside the FormItem container itself using the helpContent property.
Now, I will show you how to add help content to provide context and instruction for using a form item.

Within the Mobile Phone FormItem container and above the TextInput control, I am adding a helpContent tag block.

I am adding a Label control within the helpContent property and assigning a text property with a value of (ex. 555-555-5555).

Below the Label control, I am adding a Button control with a label property that has a value of ?, a width property that has a value of 30, and an x property with a value of 120.

I am saving the file and running the application.

You can see that the Label and Button controls have been added to the form, but notice that neither control lines up on the same horizontal baseline as the Mobile Phone field.

Back in the VehicleRequestForm file, I am adding the baseline property to both the Label and Button controls and giving them a value of 24.

When I save the file and run the application, you can see that the Label and Button controls are now aligned properly with the Mobile Phone field.

So far, neither the required icon nor the helpContent property has provided any validation to the form.

Note that, since the content in the helpContent property is simply a Button control, you can handle the click on it as you would handle any other

button click.

Of course, you can place other content in this helpContent property for display.

You will learn how to style these elements in a later video.

Actual validation in a Flex application is performed using the Flex framework Validator subclasses, which are listed here.

There are both MX and Spark validators that are implemented similarly; however, you should use the Spark validators wherever possible.

Form validation ensures that data meets specific criteria before the application uses it.

You use one of these classes to validate that the data entered into the form controls are acceptable, before they are submitted.

While the validator classes do affect the visual display of validation messages on form controls, they are not actually visual objects themselves.

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

Generally, you will create one validator for each form control.

You can give a validator an instance name using the id property.

The source property requires a binding to the instance name of the object you are validating, in this case, one of the TextInput controls.

The property argument defines the name of the object's property that you will validate.

In my example, I will validate on the text property of the TextInput control since the text property contains the text value that the user has typed.

Each validator tag has a *required* property, which is a Boolean value that indicates whether a value is required in the referenced property of the source object.

Notice that the default is true and that this is a different implementation than the required property of the FormItem containers.

That property only added the asterisk to the display.

This required property for the validators will actually ensure that users enter a value.

Each validator subclass has a set of properties that allow you to turn on specific validation rules.

The PhoneNumberValidator class specifically validates for phone numbers.

I am clicking on the Properties link in the upper right corner of the ASDocs so that you can see the validation properties that are supported by this class.

I can use the minDigits property to indicate the minimum number of digits users can type into the text field.

I can also use the allowedFormatChars value to indicate which characters I want to allow in the text fields.

Note that the default allowable characters in a PhoneNumberValidator instance are the parentheses, a dash, a period and a plus sign.

Note that the TextInput control for the Mobile Phone field has an id property of mobilePhone.

I am locating the Declarations block in my sample file and placing my cursor after the DateTimeFormatter instance.

I am adding a PhoneNumberValidator instance with an id property of mobileValidator.

Now I am adding the source property binding to the value of mobilePhone, which is the TextInput control's instance id.

Remember that, by default, if you add a validator to a field, then the field is automatically required, so you do not need to add the required property.

I need to tell the application to validate on the text property of the TextInput control instance so I am adding the property *property* with a value of text.

You can define a validator to execute based on a specific event on a specific object.

First you add the trigger property to the validator with a binding to the instance name of the object that will trigger the validation.

Then you add the triggerEvent property to the validator to name the event, on the trigger, that will trigger validation.

This form has a Button control with a label value of Submit Request and an id of submitButton.

In the validator instance, I am adding the trigger property with a binding to that submitButton instance.

Now I am adding the triggerEvent property set to the click event of the Button control.

This tells the validator to perform the validation on the mobilePhone instance's text property when the user clicks on the submitButton instance.

I noticed that I forgot the binding curly braces around the mobilePhone value in the source property.

I’m adding that now and then saving the file.

When I run the application and click on the submit Button control, you see a red border appear around the Mobile Phone TextInput control.

Also notice that the asterisk has been replaced by the default error indicator.

In addition, the help content has been replaced with a validation message that states that the field is required.

You can also trigger validation with ActionScript instead of MXML commands.

Each validator class has a validate() method that can trigger the validation.

The Validator class also has a static validateAll() method that you can use to trigger validation on multiple Validator instances at once.

Static methods can be called directly on the class rather than on the instance of a class.

Two benefits for using programmatic validation are that you have more precise control over which validation rules are run and the validation methods will return a successful or unsuccessful indication that you can then use to take other actions, like sending the form data to the server.

To use programmatic validation, you must first disable automatic validation by removing the validator’s trigger property and setting the triggerEvent property to a blank string.

Since there is no event with a blank string as a name, setting the triggerEvent property to a blank string means that automatic validation never occurs.

The validate() method of a validator class is used to trigger validation on a single validator object.

The benefit of using it over the MXML class is that it returns a ValidationResultEvent object

which has a type property that returns one of two constants:

ValidationResultEvent.VALID

and

ValidationResultEvent.INVALID.

At the end of the Script block, I am creating a private function named validatePhones that takes no parameters and does not return any value, as indicated with the void return type.

Within the validatePhones() function, I am creating a local variable, that I'm naming event, that is data typed as a ValidationResultEvent object.

Since I used content assist to assign the data type, Flash Builder generates the corresponding import statement.

I am assigning the event variable to call the mobileValidator.validate() method.

I am locating the submitButton instance in the form and setting its click handler to the method I just created, validatePhones().

I am saving the file and running the application.

When I click on the submit button, the validation on the Mobile Phone TextInput control runs exactly as it did before.

I have been validating the mobile phone form field, but there is another field that also needs to be validated.

I can use the Validator class' static validateAll() method to trigger validation on multiple validator objects at the same time.

I will pass an array of validator instances to the method as its only argument and I will receive back an array of ValidationResultEvent objects, one for each validator that fails validation.

If the returned array has a length of zero, then validation has succeeded.

Here is the Office Phone container.

Note that I have given its TextInput controls id a value of phone.

In the Declarations tag block I am adding a PhoneNumberValidator instance with an id property of officePhoneValidator.

I am also adding the source property bound to the phone TextInput control, the property property set to text and the triggerEvent property set to a blank string.

In the Script block I am adding an import statement for the mx.validators.Validator package.

Within the validatePhones() function, I am deleting the event variable and its value and replacing them with a variable named validationArray that is data typed to the Array class and set equal to the Validator.validateAll() static method.

Remember that static methods are called on the class instead of on the instance of a class.

This is why I am referencing the Validator class in this case, rather than a specific validator instance.

Within the call to the Validator.validateAll() method, I am adding array brackets to create an array of validators.

I am referencing the mobileValidator and the officePhoneValidator instances.

I am saving the file and running the application.

I am selecting an employee from the DropDownList control and removing the last digit from the Office Phone number.

When I click the Submit Request button, you can see that both TextInput controls turn red and a message displays next to both fields.

Notice that the messages are different.

The message for the office phone field indicates that the telephone number must contain at least 10 digits, while the message for the mobile phone field states that the field is required.

For your next step, work through the exercise titled “Validating form data”.