Validation

Struts 2 validation is configured via XML or annotations. Manual validation in the action is also possible, and may be combined with XML and annotation-driven validation.

Validation also depends on both the `validation` and `workflow` interceptors (both are included in the default interceptor stack). The `validation` interceptor does the validation itself and creates a list of field-specific errors. The `workflow` interceptor checks for the presence of validation errors: if any are found, it returns the "input" result (by default), taking the user back to the form which contained the validation errors.

If we're using the default settings *and* our action doesn't have an "input" result defined *and* there are validation (or, incidentally, type conversion) errors, we'll get an error message back telling us there's no "input" result defined for the action.

CONTENTS

Using Annotations

Annotations can be used as an alternative to XML for validation.

Bean Validation

With struts 2.5 comes the Bean Validation Plugin. That is an alternative to the classic struts validation described here. See the Plugin Page for details.

Examples

In all examples given here, the validation message displayed is given in plain English - to internationalize the message, put the string in a properties file and use a property key instead, specified by the 'key' attribute. It will be looked up by the framework (see Localization).

1. Basic Validation
2. Client-side Validation
3. AJAX Validation
4. Using Field Validators
5. Using Non Field Validators
6. Using Visitor Field Validator
7. How do we repopulate controls when validation fails (FAQ entry)

Bundled Validators

Note
When using a Field Validator, Field Validator Syntax is **ALWAYS** preferable than using the Plain Validator Syntax as it facilitates grouping of field-validators according to fields. This is very handy especially if a field needs to have many `field-validators` which is almost always the case.

1. conversion validator
2. date validator
3. double validator
4. email validator
5. expression validator
6. fieldexpression validator
7. int validator
8. regex validator
9. required validator
10. requiredstring validator
11. short validator
12. stringlength validator
13. url validator
14. visitor validator
15. conditionalvisitor validator

Registering Validators

{snippet:id=javadoc|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}The following list shows the default validators included in the framework and is an example of the syntax used to declare our own validators.{snippet:lang=xml|url=struts2/core/src/main/resources/com/opensymphony/xwork2/validator/validators/default.xml}

Struts 2.1 and Prior
The `validators.xml` used to reference a DTD hosted by Opensymphony, the original location of the XWork project. Since the the move to Apache Struts, DTDs were changed. Please ensure in your projects to include the DTD header as described in the examples found here
Struts 2.0.7 and Prior
The `validators.xml` containing custom validators needs to contain a copy of the default validators. No DTD was used in validators.xml. See: http://struts.apache.org/2.x/docs/release-notes-208.html#ReleaseNotes2.0.8-MigrationfrompreviousReleases

Turning on Validation

The default interceptor stack, "defaultStack", already has validation turned on. When creating your own interceptor-stack be sure to include **both** the `validation` and `workflow` interceptors. From `struts-default.xml`:

xml<interceptor-stack name="defaultStack"> ... <interceptor-ref name="validation"> <param name="excludeMethods">input,back,cancel,browse</param> </interceptor-ref> <interceptor-ref name="workflow"> <param name="excludeMethods">input,back,cancel,browse</param> </interceptor-ref> </interceptor-stack>

Beginning with version 2.0.4 Struts provides an extension to XWork's `com.opensymphony.xwork2.validator.ValidationInterceptor` interceptor.

xml<interceptor name="validation" class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/>

This interceptor allows us to turn off validation for a specific method by using the `@org.apache.struts2.interceptor.validation.SkipValidation` annotation on the action method.

Validator Scopes

{snippet:id=fieldValidators|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=nonFieldValidators|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}

Notes

{snippet:id=validatorsNote|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}

Defining Validation Rules

{snippet:id=validationRules1|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=exValidationRules1|lang=xml|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=validationRules2|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}

In this context, "Action Alias" refers to the action name as given in the Struts configuration. Often, the name attribute matches the method name, but they may also differ.

Localizing and Parameterizing Messages

{snippet:id=validationRules3|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=exValidationRules3|javadoc=true|lang=xml|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=validationRules4|javadoc=true|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}{snippet:id=exValidationRules4|javadoc=true|lang=xml|url=com.opensymphony.xwork2.validator/ValidatorFactory.java}

Validator Flavor

{snippet:id=validatorFlavours|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}

Non-Field Validator Vs Field-Validator validatortypes

There are two ways you can define validators in your -validation.xml file:

1. <validator>
2. <field-validator>

Keep the following in mind when using either syntax:

Non-Field-Validator: The <validator> element allows you to declare both types of validators (either a plain Validator a field-specific FieldValidator).

xml<validator type="expression"> <param name="expression">foo gt bar</param> <message>foo must be great than bar.</message> </validator> xml<validator type="required"> <param name="fieldName">bar</param> <message>You must enter a value for bar.</message> </validator>

field-validator: The <field-validator> elements are basically the same as the <validator> elements except that they inherit the fieldName attribute from the enclosing <field> element. FieldValidators defined within a <field-validator> element will have their fieldName automatically filled with the value of the parent <field> element's fieldName attribute. The reason for this structure is to conveniently group the validators for a particular field under one element, otherwise the fieldName attribute would have to be repeated, over and over, for each individual <validator>.

It is always better to defined field-validator inside a <field> tag instead of using a <validator> tag and supplying fieldName as its param as the xml code itself is clearer (grouping of field is clearer)
Note that you should only use FieldValidators (not plain Validators) within a block. A plain Validator inside a <field> will not be allowed and would generate error when parsing the xml, as it is not allowed in the defined dtd (xwork-validator-1.0.2.dtd)

Declaring a FieldValidator using the <field-validator> syntax:

xml<field name="email_address"> <field-validator type="required"> <message>You cannot leave the email address field empty.</message> </field-validator> <field-validator type="email"> <message>The email address you entered is not valid.</message> </field-validator> </field>

The choice is yours. It's perfectly legal to only use elements without the elements and set the fieldName attribute for each of them. The following are effectively equal:

xml<field name="email_address"> <field-validator type="required"> <message>You cannot leave the email address field empty.</message> </field-validator> <field-validator type="email"> <message>The email address you entered is not valid.</message> </field-validator> </field> <validator type="required"> <param name="fieldName">email_address</param> <message>You cannot leave the email address field empty.</message> </validator> <validator type="email"> <param name="fieldName">email_address</param> <message>The email address you entered is not valid.</message> </validator>

Short-Circuiting Validator

{snippet:id=shortCircuitingValidators1|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}{snippet:id=exShortCircuitingValidators|lang=xml|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}{snippet:id=shortCircuitingValidators2|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}{snippet:id=scAndValidatorFlavours1|1=javadoc|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}{snippet:id=exScAndValidatorFlavours|lang=xml|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}{snippet:id=scAndValidatorFlavours2|1=javadoc|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}

How Validators of an Action are Found

{snippet:id=howXworkFindsValidatorForAction|javadoc=true|url=com.opensymphony.xwork2.validator/Validator.java}

Writing custom validators

If you want to write custom validator use on of these classes as a starting point:

- com.opensymphony.xwork2.validator.validators.ValidatorSupport
- com.opensymphony.xwork2.validator.validators.FieldValidatorSupport
- com.opensymphony.xwork2.validator.validators.RangeValidatorSupport
- com.opensymphony.xwork2.validator.validators.RepopulateConversionErrorFieldValidatorSupport

Resources

WebWork Validation

Next: Localization