Using Form Validation (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introduction

For most web applications input is essential. Apache Cocoon provides variety of modules to support basic interaction like simple syntax checking of input data or writing input data to databases.

A new but already very mature modules is the Cocoon Forms (former Woody) framework. Another, simpler and older implementation is the validation using the FormValidatorAction, which is described here.

To validate user input, an action, the FormValidatorAction, is placed in your pipeline. Together with a descriptor file, that specifies the allowed input, this action validates the request parameters. Based on the result, a different page can be displayed and feedback can be given to the user either using XSP and the formval logicsheet or the SimpleFormTransformer.

1.1. Sitemap Usage

To take advantage of the form validator action, create two pages. One for the input form and one indicating the acceptance of the reservation. Create a pipeline in your sitemap so that the confirmation page is only shown when the action completed successfully and the input form is returned otherwise.

```
<?xml version="1.0"?> <map:match pattern="car-reservation"> <map:act
type="form-validator"> <!-- add you favourite database action here --> <map:parameter
name="descriptor" value="descriptor.xml"/> <map:parameter name="constraint-set"
value="car-reservation"/> <map:generate type="serverpages" src="OK.xsp"/>
<map:transform src="stylesheets/dynamic-page2html.xsl"/> <map:serialize/> </map:act>
<map:generate type="serverpages" src="test/ERROR.xsp"/> <map:transform
src="stylesheets/dynamic-page2html.xsl"/> <map:serialize/> </map:match>
```

Note here that you may not use a redirection to point to the pages if you would like to access the validation results e.g. on the error page. A redirection would create a new request object and thus discard the validation results.

A different example, that does not need serverpages but the SimpleFormTransformer:

```
<?xml version="1.0"?> <map:match pattern="car-reservation"> <map:act
type="req-params"> <map:parameter name="parameters" value="order"/> <map:act
type="form-validator"> <map:parameter name="descriptor" value="descriptor.xml"/>
<map:parameter name="constraint-set" value="car-reservation"/> <!-- add you favourite
database action here --> <map:generate type="file" src="OK.xml"/> <map:transform
src="stylesheets/dynamic-page2html.xsl"/> <map:serialize/> </map:act> </map:act>
<map:generate type="file" src="test/ERROR.xml"/> <map:transform
src="stylesheets/dynamic-page2html.xsl"/> <map:transform type="simple-form"/>
<map:serialize/> </map:match>
```

Although this looks more complicated at first, it has advantages if you don't want to or cannot use XSP. For example, if the form is stored as XHTML in a database, XSP could not be used to fill the form with values from request parameters or to display detailed error messages.

Keep in mind that files, here the descriptor file, could be specified using the cocoon: pseudo-protocol. Thus the file could be generated dynamically from another pipeline!

1.2. The Descriptor File

For details on the syntax of the descriptor file see javadocs. Basically it consists of two sections, a list of parameters and their properties and a list of constraints or constraint sets. The file syntax is set up so that it can be shared with the database actions.

1.2.1. The types recognized by validator and their attributes

string	nullable="yes|no" default="str"
long	nullable="yes|no" default="123123"
double	nullable="yes|no" default="0.5"

Default value takes place only when specified parameter is nullable and really is null or empty. Long numbers may be specified in decimal, hex or octal values as accepted by java.Lang.decode (String s).

1.2.2. Constraints

matches-regex	POSIX regular expression
one-of	List of strings, enclosed and separated by |
min-len	positive integer
max-len	positive integer
min	Double / Long
max	Double / Long

Constraints can be defined globally for a parameter and can be overridden by redefinition in a constraint-set. Thus if e.g. a database field can take at maximum 200 character, this property can be set globally.

Values in parameter arrays are validated individually and the worst error is reported back.

```
<?xml version="1.0"?> <root> <parameter name="persons" type="long" min="1" default="4" nullable="no"/> <parameter name="deposit" type="double" min="10.0" max="999.99"/> <parameter name="email" type="string" max-len="50" matches-regex="^[\d\w][\d\w\-_\.]*@([\d\w\-_]+\.)\w\w\w?$"/> <parameter name="colour" type="string" one-of="|red|green|blue|white|"/> <constraint-set name="car-reservation"> <validate name="persons"/> <validate name="deposit" min="50.0"/> <validate name="email"/> </constraint-set> </root>
```

The above could for example describe expected input from a reservation form. Specifications in a constraint set take precedence over the general ones.

1.3. XSP Usage

To give the user some feedback why her/his submitted data was rejected there is a special taglib "xsp-formval". Declare its name space as usual.

If only interested in validation results, just:

```
<xsp-formval:on-ok name="persons"> <myapp:error>(ERROR)</myapp:error> </xsp-formval:on-ok>
```

Alternatively, if you just want a boolean value from the logicsheet if a test is successful, use this method:

<xsp:logic> if (!<xsp-formval:is-ok name="persons"/>) { <myapp:error>(ERROR)</myapp:error> }; </xsp:logic>

Internationalization issues are a separate concern and are not discussed here.

Currently the following validation result codes are supported:

tag	Meaning
xsp-formval:is-ok	no error occurred, parameter successfully checked
xsp-formval:is-error	some error occurred, this is a result that is never set but serves as a comparison target
xsp-formval:is-null	the parameter is null but isn't allowed to
xsp-formval:is-toosmall	either value or length in case of a string is less than the specified minimum
xsp-formval:is-toolarge	either value or length in case of a string is greater than the specified maximum
xsp-formval:is-nomatch	a string parameter's value is not matched by the specified regular expression
xsp-formval:is-notpresent	this is returned when the result of a validation is requested but no such result is found in the request attribute

For debugging purposes or if you would like to iterate over the validation results, xsp-formval:results returns a java.util.Map containing them all.

If you would like to be more specific what went wrong, you can query the descriptor file for attributes.

First set the url of the file or resource that contains the parameter descriptions and constraint sets. This needs to be an ancestor to all other tags (of this taglib). Multiple use of this tag is allowed (although probably not necessary).

You need to do this only if you plan to query the descriptor file or if you'd like to use the shorthand below.

<xsp-formval:descriptor name="descriptor.xml" constraint-set="reservation"> deposit must be at least EUR <xsp-formval:get-attribute parameter="deposit" name="min"/> </xsp-formval:descriptor>

If you need to use one parameter a lot, there's a short hand. Use this e.g. if you'd like to set up the properties of an input tag according to the information from the descriptor file or if you'd like to give detailed error messages.

Note that you can specify additional attributes in the description file that are not understood (and therefore ignored) by the FormValidatorAction but that could be queried here. This might be e.g. the size of the input field which might be different from the max-len a parameter can take.

<xsp-formval:descriptor name="descriptor.xml" constraint-set="car-reservation"> <xsp-formval:validate name="deposit"> <xsp:logic> if (<xsp-formval:is-null/>) { <myapp:error> (you must specify a deposit)) </myapp:error> } else if (<xsp-formval:is-toosmall/>) { <myapp:error> (deposit is too small (<

<xsp-formval:get-attribute name="min"/>)) </myapp:error> } else if (<xsp-formval:is-toolarge/>) { <myapp:error> (deposit is too large (> <xsp-formval:get-attribute name="max"/>)) </myapp:error> } else { <myapp:error> (ERROR) </myapp:error> }; </xsp:logic> </xsp-formval:validate> </xsp-formval:descriptor>

1.4. SimpleFormTransformer

An alternative solution to using the formval logicsheet and XSP is to use the SimpleFormTransformer. If fills the form with values obtained from request parameters, overwriting existing values. Hence the data entered by the user is not discarded when it does not validate successfully.

Beware when using the SimpleFormTransformer together with XSP: The observer behaviour can be very confusing when trying to set a value from XSP and it is silently overwritten by the transformer!

When a form element carries the attribute fixed="true", the transformer does not replace the value.

Feedback can be given to the user through <error/> tags. Error tags need to have a name attribute identical to the input element they refer to. Multiple error elements may be present for any input element. The FormValidatorAction sets a special field * that indicates whether all parameters were validated successfully or not.

An error element is omitted together with all contents whenever the specified condition is not met. Conditions are either exact or greater equal constraints indicated by the attribute when or when-ge respectively.

Allowed values for error conditions are: ok, not-present, error, is-null, too-small, too-large, no-match

<input name="email" type="text"/><error name="email" when-ge="error">*</error> <!-- ... --> <error name="email" when="is-null">Please enter your email address.</errro> <error name="email" when="no-match">Please enter a syntacticly correct email address.</errro>

1.5. Other Validations

In addition to validating form input, other actions exists that validate values from different sources using the same techniques and syntax. For example, the SessionValidatorAction operates on session attributes.

1. Comments

add your comments