ValidatorSetup

This is a basic 1,2,3 guide that anyone can expand upon when they have something useful to add:\

- get commons-validator.jar - go to jakarta to download the latest version.
- put it in your classpath - this is normally WEB-INF/lib/ along with the other jars such as struts.jar
- create action forms to be validated - based on ValidatorForm, or just use DynaActionForms and don't worry about coding classes for them - all that's needed is the form config in struts-config.xml (see below)
- edit your struts-config.xml:
    - put the validator plug-in xml into the plug-ins section:

```
<!-- sets up Validator -->
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

- create the form configuration in struts-config.xml for the form-names, e.g. for ValidatorForms:

```
<form-bean name="surveyForm"
  type="com.mystuff.myapp.myForm">
</form-bean>
```

- or for DynaValidatorForms (we have to specify the fields we want here - and it's best to stick to Strings rather than trying to deal with Integers or Dates etc which do not get populated if the user enters the wrong type of data):

```
<form-bean name="surveyForm"
  type="org.apache.struts.validator.DynaValidatorForm">
  <form-property name="surveyForm"
    type="java.lang.String"/>
  <form-property name="title"
    type="java.lang.String"/>
</form-bean>
```

- configure the action mapping e.g.:

```
<action path="/private/esurvey/survey/update"
  type="org.gargantus.esurvey.SurveyAction"
  name="surveyForm"
  scope="request"
  validate="true"
  parameter="save"
  input="/private/esurvey/survey/validate/failed.do"
  roles="user">
  <set-property property="secure" value="true"/>
  <forward name="failed" path=".survey"/>
  <forward name="display" path=".survey"/>
  <forward name="finished"
    path="/private/esurvey/survey/list.do"
    redirect="true"/>
</action>
```

- set validate=true
- set input=/my/path so that struts can send the request somewhere on validation-failure - this can be another action path like the example above, or a tiles definition (normally beginning with '.'), or any other page in your app.
- add form-name as 'name' attribute
- set the scope to request, or if you want to make use of 'work-flow' or 'wizard' techniques or other patterns that allow you to accumulate data over several HTTP requests, put 'session' instead.

- create validation.xml config file in WEB-INF - using the same form names as in struts-config.xml e.g.:

```
<form-validation>
  <formset>
    <form name="surveyForm">
      <field property="surveyId"
        depends="required,integer">
        <arg0 key="esurvey.survey.validate.id"/>
      </field>
      <field property="dateLive"
        depends="date">
        <arg0 key="esurvey.survey.validate.dateLive"/>
        <var>
          <var-name>datePattern</var-name>
          <var-value>yyyy-MM-dd</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>
```

- This validation.xml snippet is up-to-date as of validator-1.1, but changes are in the pipeline for localization, resource bundles etc and some xml attributes may be deprecated.
- get validator-rules.xml from the struts installation, in the conf/share/ directory and put it in WEB-INF
- and now on to the JSPs. First of all, none of the tags will work unless you have their taglib declaration at the top of your page so:

```
<%@ taglib prefix="html" uri="http://jakarta.apache.org/struts/tags-html" %>
```

- use html:form in the JSPs whose forms will be validated, e.g.:

```
<html:form action="/private/esurvey/survey/update"
  focus="optionText">
```

- to enable client-side validation:
  - make sure you include the onsubmit attribute with the html:form tag

```
<html:form action="/private/esurvey/survey/update" >
  focus="optionText"
  onsubmit="return validateForm(this);">
```

- include in the JSP the html:javascript tag with the same form-name as above and dynamic=true

```
<html:javascript formName="surveyForm"
  method="validateForm"
  dynamicJavascript="true"
  staticJavascript="false"
  cdata="false" />
```

- the 'method' should be the same as what you specified for the html:form onsubmit attribute.
- you could set static=true there as well, but it is more efficient to create an extra page with just that taglib in it, set to produce only static javascript (needs no form-name then) - and refer to it from your JSP using <script> tags. This way, the browser will cache it and reduce traffic.

```
<script src="<html:rewrite page="/staticjavascript.do"/>"
  type="text/javascript"></script>
```

- use the messages or the errors tag to display the validation errors (this has got JSTL tags too) (PS is the errors tag still in use?)

```
<html:messages id="validatorMsgs"
          message="false"
           header="errors.header"
           footer="errors.footer">
  <p>
    <c:out default="default text for &lt;c:out&gt; error in validatorMsgs"
           value="${validatorMsgs}"
       escapeXml="false"/>
  </p>
</html:messages>
```

And finally, this is how struts actually deals with it all on submission of a form:

1. It goes and checks for any instances of the form-bean in the scope (the name of the class, type, and scope is specified by the action mapping in struts-config.xml).
2. If found then call the form's reset method is called.
3. If not found than a new instance is created and stored in the proper scope.
4. Then the formBean is populated with request parameters by struts
5. If the validate is true than the form's validate() method is called. There's no need to call it explicitly and because of ValidatorForm doing all the validation automatically, all that is required for code is a call in your form's submit() method to super.validate(). If you are using DynaValidatorForm , then you will not have program any form beans at all.
    a. if validate() returns any errors, then struts forwards the request to the URL specified in the input attribute of the action mapping.
    b. otherwise it checks for the action which the action mapping is based on.

Then after that you have a populated form. Normally you would then transfer the data into a value or data-transfer object to pass to your business layer (see BeanUtils.copyProperties).