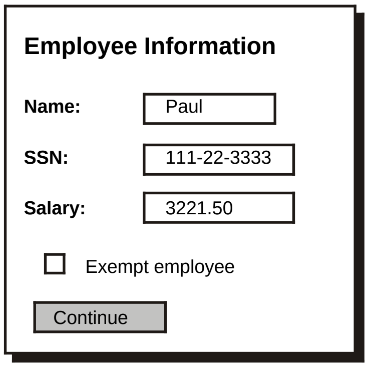
Struts Validation

- What is Validation?
- Client-Side and Server-Side Validation
- Struts Validation Techniques

What is Validation?

- Validation lets an application verify that users enter proper data, for example that a serial number conforms to a particular pattern or that a zip code is numeric
- Validation is required for real-world Web applications not only so that the application functions properly but also to prevent attacks (e.g. buffer overflow or SQL injection attacks)



All programs, even standalone programs, need to check the input from users, but validation is especially critical for Web applications. That's because Web apps involve network traffic and are susceptible to attacks by "script kiddies" and others interested in cracking or breaking the system.

So all well designed Web application should check user input, but note that you really only need to check user input widgets where the user enters data. For example, in the figure on this page, the Web app should validate the Name, SSN and Salary fields, but doesn't need to vet the "Exempt employee" checkbox data.

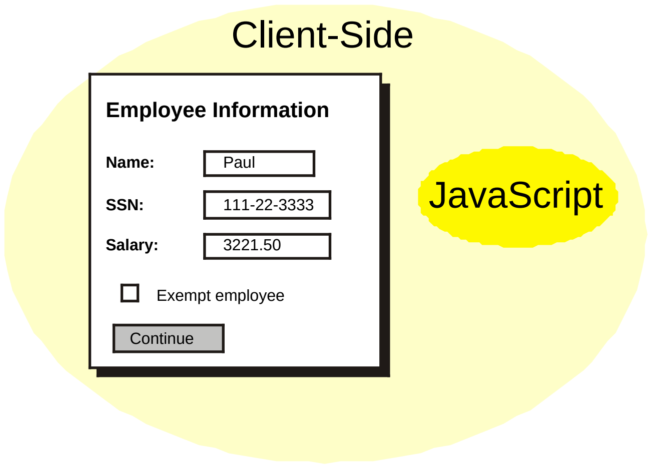
Validation Scenarios

Validation	Description
Required Field	Ensure that user entered data in the field
Data Type	Ensure that user enter correct kind of data (e.g. an integer)
String Length	Ensure that the user entered at least n characters but less than m
Value Range	Ensure that numeric values fall within a range
Pattern Match	Ensure that entered text matches a regular expression

This page lists some of the generic kinds of validation performed by Web applications. Note that it doesn't include application-specific types of validation, for example to verify that an entered serial number actually matches a real employee. We can perform application-specific validation in Struts-based applications, but that will require writing code.

Using Client-Side Validation

- Client-side validation uses scripting in the browser to check data before the form is submitted to the server
- Client-side validation makes the application more responsive, but does not replace server-side validation because the user can disable browser scripting



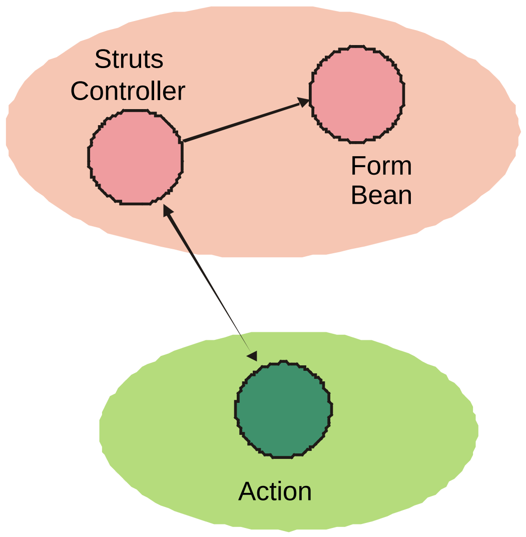
Client side validation requires that the user's browser support scripting (Struts uses JavaScript). The main advantage of client-side validation is that since the checking is done entirely on the client, you save network bandwidth.

The big gotcha with client-side validation is threefold: First, not all browsers fully support JavaScript. Second, users can disable JavaScript in the browser (usually to avoid popups and the like). Third, JavaScript programs running on the client cannot access server resources, such as databases.

It's also important to note that savvy hackers can easily avoid client-side validation and therefore bypass its checking. Therefore, you should never solely depend on client-side validation.

Using Server-Side Validation

- Server-side validation is performed by the Web application running on the server – it can access server resources such as databases to perform the validation

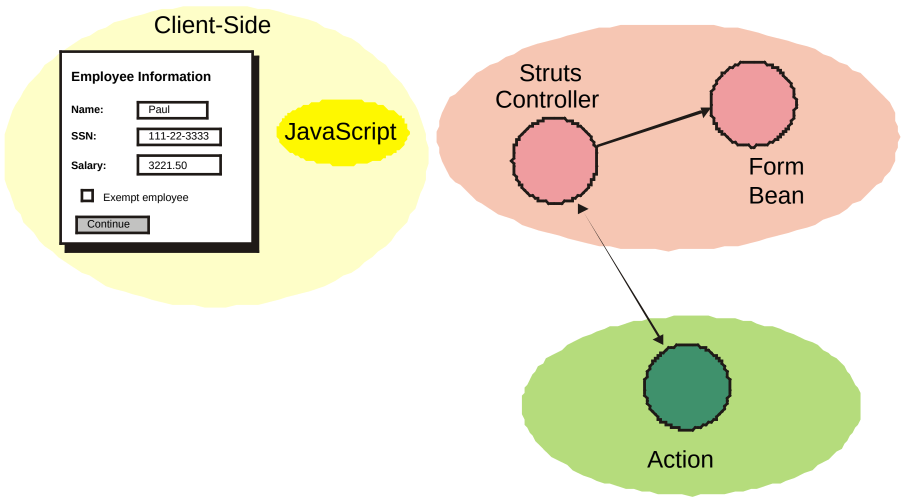


Server-side validation requires the Web application to retrieve data from the request and examine the data for correctness. Since the code is running on the server, it can read databases, access directory services and so forth as needed to perform the validation.

The negative for server-side validation is that it occupies server CPU cycles and thus affects scalability. In addition, it requires network bandwidth to pass the request to the server and then perform some sort of "post back" to redisplay the offending page if validation fails.

Client-Side vs Server-Side Summary

- Because of the strengths and weaknesses of the techniques, Web applications can use both client and server-side validation
- Note that server-side is always required, even if the also validates

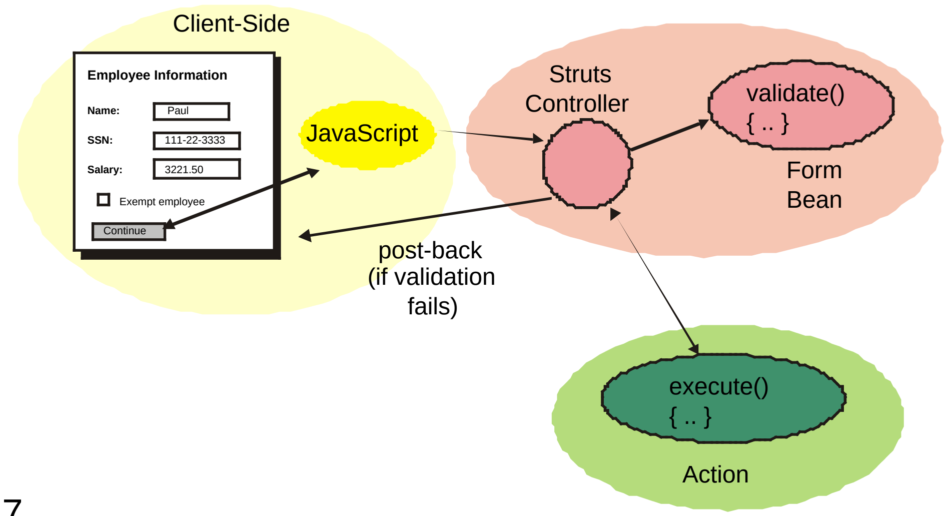


Because client-side validation is unreliable and generally simplistic, it's absolutely required that all well written Web applications perform validation on the server, even if you have enabled client-side validation.

So why bother with client-side validation at all? To save network bandwidth and make your Web application more responsive by avoiding unnecessary roundtrips.

Struts Validation Techniques

- Programmatic validation in the ActionForm (server-side)
- Declarative validation using the Struts Validator framework (client and server-side supported)
- Programmatic validation in the Action (server-side)



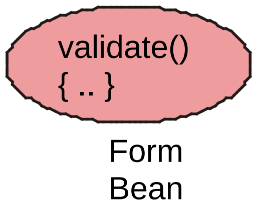
Struts provides several techniques you can use to validate user input, both on the client and on the server.

In Struts version 1.0, most programs used only server-side validation using a combination of code in the ActionForm's validate() method and the Action's execute() method. Struts version 1.1 still supports those techniques, but provides another option: declarative validation, where you configure the validation rules and Struts generates the validation code for you (both JavaScript and in the ActionForm).

Note that if you decide to validate in the Action's execute() method, you will need to define a mapping so that the Struts controller transfers control back to the offending page is validation fails. The act of re-displaying a page with validation errors is referred to as a "post-back". The Struts controller automatically does a post-back if validation fails in the ActionForm.

Programmatic ActionForm Validation

- This technique has been in Struts since the beginning – you override the validate() method in the ActionForm and add errors to an error collection
- If the error collection is non-empty, the Struts controller performs a *postback* to the input form which can display the errors to the user with Struts HTML tags
- For maximum flexibility, you can define the error messages in an ApplicationResources.properties file instead of inline in the code



The ActionForm is a nice place from which to perform validation. The ActionForm superclass provides a method named validate() that you can override and return a collection of error objects, each of which contains an error string. If validation fails (i.e. the error collection has entries), then the offending form is re-invoked. The form can then display the error messages, giving the user an opportunity to try again.

Programmatic ActionForm Validation, cont'd

- The validate() method can check string length, data types and so forth

```
public ActionErrors validate ( ActionMapping mapping
                             , HttpServletRequest request)
{
  ActionErrors errors = new ActionErrors();

  if ( ( name == null ) || name.trim().length() == 0 )
    errors.add ( "name"
          , new ActionError ( "name.required" ) );

    . . .

  return errors;
}
```

ApplicationResources.properties

```
name.required=Please enter a name

. . .
```

Here we show a fragment of an ActionError class's validate() method. It's important to note that the Struts controller calls validate() after it calls the setXxx() methods to populate the ActionForm with the user's input. So the validate() method can check its own properties (here, the "name" property is shown) and add an ActionError object to a collection if validation fails.

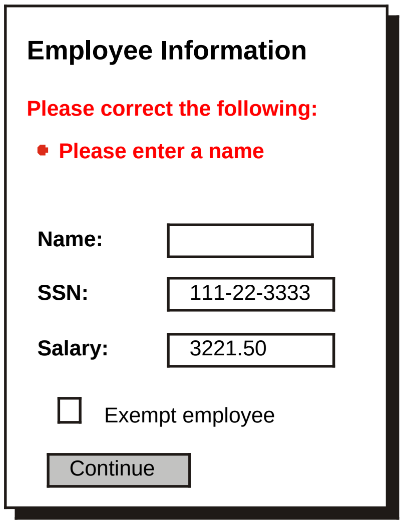
In this case, we are ensuring that the "name" property has at least some characters in it.

To avoid hard-coding error strings, we instead return an ActionError object that contains a keyed error string. The Struts controller uses the key to look up the actual error message string from a properties file – that way, if we want to change the message (perhaps translate to a different national language), we don't need to recompile the ActionForm.

Displaying Validation Errors

- Struts provides custom actions in the *logic* and *html* libraries to format and display validation error message

```
<logic:messagesPresent>
  <font color="red">
    <p>
      Please correct the following:
    </p>
    <ul>
      <html:messages id="error">
        <li>
          <c:out value="${error}"/>
        </li>
      </html:messages>
    </ul>
  </font>
</logic:messagesPresent>
```



Employee Information

Please correct the following:

- **Please enter a name**

Name:	
SSN:	111-22-3333
Salary:	3221.50

☐ Exempt employee

When the Struts controller performs a post-back, it stores the list of error messages in the request (or session) so that the HTML "errors" custom action can display them.

Theses Struts 1.1 custom actions provide a lot of flexibility in formatting and displaying errors (Struts 1.0 was much more limited).

Programmatic Action Validation

- In some cases, you may wish to perform validation in the model rather than the controller (the ActionForm is a Controller component)

```
public ActionForward execute (
    . . .ActionForm form . . . )
{
    EmployeeForm theForm =
        (EmployeeForm)form;

    if ( !isSsnOK (theForm.getSsn() )
        return mapping.findForward
          ( "validation-failed" );
    . . .
}
```

Here we show a fragment of the execute() method in an Action. It calls an application-specific isSsnOK() method which presumably accesses a database to see if the entered social-security number matches the number of an employee. If not, the Action returns a mapping string which causes the Struts controller to transfer control to some sort of error page.

Note that you could be more clever and have the action store errors in the request/session in the same way that the Struts controller does for ActionForm errors. See the Struts documenation for details.

Configuring Validation

- In the struts-config.xml file, you configure validation for a given action by setting the validate attribute to "true"

```
<action-mappings>
  <action
   path="/EmployeeInfo"
   type="actions.UpdateEmployee"
   scope="request"
   name="UpdateEmployeeForm"
   input="/update-employee.jsp"
   unknown="false"
   validate="true">
   <forward name="success"
     redirect="false"
     path="/index.jsp"/>
   <forward name="validation-failure"
     redirect="false"
     path="/error.jsp"/>
  </action>
  . . .
</action-mappings>
```

Post-back location

Configures validation

Returned by Action if validation fails

Here we show a fragment of the struts-config.xml file that configures a Struts application. Especially note the "validate" attribute and the "input" attribute, which is where the Struts controller "posts-back" if the ActionForm returns a non-empty error collection.

The Struts Validation Framework

- The Struts Validation framework lets you perform validation declaratively (without writing code)
- The framework is bundled with Struts version 1.1 and later and uses the Jakarta Commons Validation framework
- You configure the framework by writing XML descriptions of the desired validations

Programmatic validation is fine, but it gets tedious and repetitive for standard scenarios. The Struts Validation Framework lets you avoid the tedium – instead of writing code, you declare your validation requirements and Struts takes care of it. This carries on the idea introduced with the dynamic action-forms we learned about in an earlier chapter.

Validations Supported by the Validation Framework

- The framework supports most of the common validation scenarios and you can extend it either by writing your own validators or by combining it with the programmatic validation techniques covered earlier

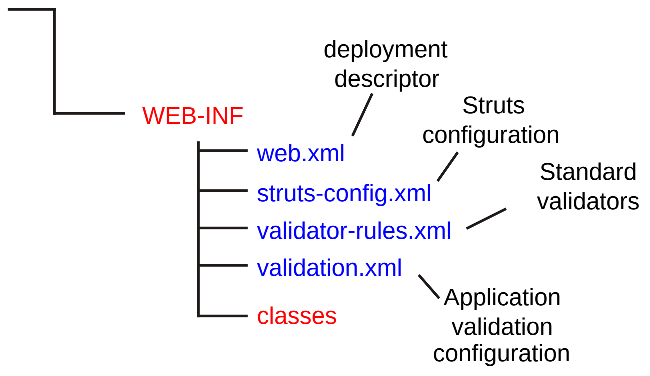
Validation	Description
Required Field	Ensure that user entered data in the field
Data Type	Ensure that user enter correct kind of data (e.g. an integer)
String Length	Ensure that the user entered at least n characters but less than m
Value Range	Ensure that numeric values fall within a range
Pattern Match	Ensure that entered text matches a regular expression

The validation-rules.xml file defines the standard validators included with the validation framework. These standard validators handle the typical scenarios we've touched on in this chapter. In addition, you can combine declarative validation with the programmatic validation covered earlier, so you can handle any cases not covered by the standard validators.

It's also possible to write your own custom validators, but we will not cover that topic in this course.

Configuring Struts Validation Framework

- To configure the framework, you must add create a file named validation.xml and include the Struts validation-rules.xml in the WAR file
- You must code the validation.xml file
- The validation-rules.xml file contains the definitions of the standard Struts validators
- You also must add a "plug-in" entry for the Validator in the struts-config.xml file

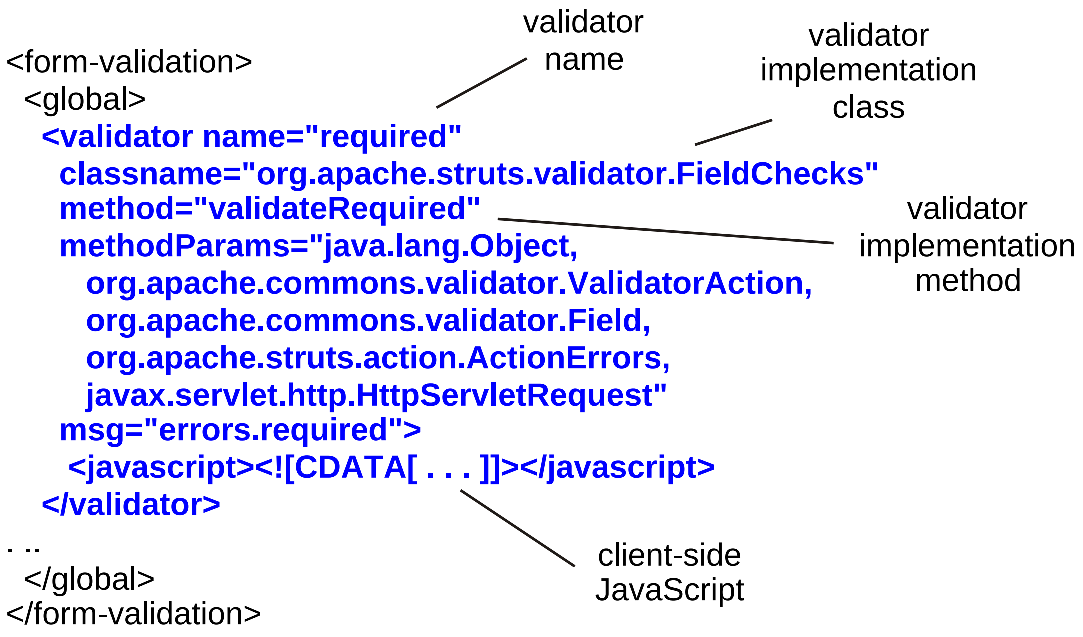


To use declarative validation, you include the standard validator-rules.xml file in your Web application and then create the validation.xml file to declare your validation requirements.

It's also important to note that the standard validators include client-side JavaScript to decrease network roundtrips and improve the responsiveness of the application.

The Validator-rules. xml File

- This file configures the standard Struts validators and defines the JavaScript used for client-side validation


validator name
validator implementation class
validator implementation method
client-side JavaScript

```
<form-validation>
  <global>
    <validator name="required"
      classname="org.apache.struts.validator.FieldChecks"
      method="validateRequired"
      methodParams="java.lang.Object,
         org.apache.commons.validator.ValidatorAction,
         org.apache.commons.validator.Field,
         org.apache.struts.action.ActionErrors,
         javax.servlet.http.HttpServletRequest"
      msg="errors.required">
       <javascript><![CDATA[ . . . ]]></javascript>
    </validator>
. ..
  </global>
</form-validation>
```

For learning purposes, it's good to know about and study the validation-rules.xml file, but you don't generally need to modify it.

Here we show one of the standard validators defined in the validation-rules.xml file – the "required" validator, which checks to see if the user entered any data into an associated field.

The "validator" element defines a validator's name, implementation class and method, arguments to the method, default error message key and optionally the JavaScript to perform client-side validation.

Like most of the standard validators, the "required" validator is actually implemented by the FieldChecks class, which is part of the Struts JAR file.

The Validator.xml File

- To use declarative validation, you must complete this file to map validators to the fields in your ActionForms

```
<form-validation>
  <formset>
   <form name="RegisterForm">
     <field property="ssn"
          depends="required,mask">
      <arg0 key="ssn.display" />
      <var>
        <var-name>mask</var-name>
        <var-value>
          [0-9]{3}-[0-9]{2}-[0-9]{4}
        </var-value>
      </var>
     </field>
     . . .
    </form>
  . . .
  </formset>
</form-validation>
```

form to validate

field to validate

validator list

keyed message string for validation failure

variables (parameters) (validator-specific)

This is the critical file when using declarative validation. In it, you assign validators to the fields (actually properties) from an input form. For each field, you use the "depends" attribute to list one or more validators, a portion of the message string, and then, depending on the validators, you supply additional information via "variables". In this example, we are configuring the "required" validator, which needs no additional information, and the "mask" validator, which requires a regular expression.

Error Message Strings

- You typically define validation error message strings in the ApplicationResources.properties file

```
#-- validation error base strings
errors.required={0} is required
errors.integer=The {0} must be a whole number
errors.range=The {0} is not in the range {1} through {
errors.invalid=The {0} is not correctly formatted
errors.email=The {0} does not appear to be valid
errors.date=The {0} does not appear to be a valid date

#-- Registration form
name.display=name
email.display=email address
age.display=age
ssn.display=Social Security number
graduationDate.display=graduation date
```

As we've already seen, for most flexibility, it's best to define message strings in the Web app's property file.

When you use declarative validation, Struts builds an individual message string from two places: a "base" string and a field-specific string. Note that the base string contains placeholders, for example {0}, where Struts will substitute the field-specific string. More on this on the next page.

Also note that base string names pretty much match validator names – the one exception is the "errors.invalid" base string, which corresponds to the "mask" validator. You can see how each of the base strings maps to a validator by examining the validator-rules.xml file.

Error Message String Construction

- Assuming that the entered Social Security number fails the *mask* validator:

validation.xml

```
<form name="RegisterForm">
  <field property="ssn"
        depends="required,mask">
    <arg0 key="ssn.display" />
    <var>
      <var-name>mask</var-name>
      <var-value>
        [0-9]{3}-[0-9]{2}-[0-9]{4}
      </var-value>
    </var>
```

validation-rules.xml

```
<validator name="mask"
    . . .
    msg="errors.invalid">
</validator>
```

ApplicationResources.properties

```
errors.invalid=The {0} is not correctly formatted
ssn.display=Social Security number
```

$\text{The } \underbrace{\text{Social Security number}}_{\{0\}} \text{ is not correctly formatted}$

The error message that the Struts controller returns on a post-back is constructed in a flexible, but perhaps complex way. When the validator returns "false", Struts first looks in the validator-rules.xml file to find the key for the base message string for that validator, in this case "errors.invalid". Struts then replaces any placeholders with the "arg0" key specified in validation.xml.

This technique lets you create generic base message strings and customize and re-use them for multiple fields.

You should also note that validator-rules.xml defines default, generic base message strings for all of the standard validators. Any entries with the same key in the ApplicationResources.properties file override the generic message strings. Generally, you'll want to provide your own base message strings to match the "voice" of your application.

Required-Field Validator

- This validator determines if a field is not null and if the length of the field is greater than zero, not including whitespace
- If the validation fails, the validator adds the specified keyed message string to the ActionErrors collection
- You define the mapping for the keyed message string in the ApplicationResources.properties file

validation.xml

```
<field property="firstName"
      depends="required">
  <arg0 key="firstName.display" />
</field>
```

ApplicationResources.properties

```
errors.required={0} is required
firstName.display=First name
```

The "required" validator is very simple – it checks to see if the text for a property has any characters. If not, the validator returns false and Struts constructs an error message string as described previously.

Often, you will use the "required" validator in combination with other validators, since the other validators generally accept empty input as OK.

Data Type Validators

- These validators determine if the text in a field is convertible to one of these standard Java types: byte, short, integer, long or double

validation.xml

```
<field property="age"
   depends="required,integer">
  <arg0 key="age.display" />
</field>
```

ApplicationResources.properties

```
errors.integer=The {0} must be a whole number
age.display=age
```

The data-type validators check to see if entered data matches the form of the specified data type.

Here we show an example of using the "integer" validator to ensure that the user enters a whole number for an "age" field".

Note that it's common to use the "required" validator in conjunction with a data-type validator, since the data-type validators accept empty input as OK.

Range Validators

- These *intRange* and *floatRange* validators determine if a field contains an integer or float within a specified range
- You can supply the range as arguments in the field definition

validation.xml

```
<field property="age"
      depends="required,integer,intRange">
  <arg0 key="age.display" />
  <arg1 name="intRange" key="${var:min}"
      resource="false"/>
  <arg2 name="intRange" key="${var:max}"
      resource="false"/>
  <var>
    <var-name>min</var-name>
    <var-value>12</var-value>
  </var>
  <var>
    <var-name>max</var-name>
    <var-value>99</var-value>
  </var>
</field>
```

ApplicationResources.properties

```
errors.range=The {0} is not in the range {1} through {2}
age.display=age
```

These validators ensure that numeric values fall within a range that you specify as "variables" in validation.xml. You also need to configure the validator to pass the range values so the validator can substitute the range values into placeholders in the base string. When defining the range arguments, you need to specify whether the arguments will come from a resource file entry or be defined in validation.xml itself. Here we choose the options ${var.min} and ${var.max} to indicate that the range values are defined as variables in validation.xml.

This example shows using the integer-range validator for an "age" field (this example is an extension of the one on the previous page). In addition to being an integer, the entered value for "age" must be greater than or equal 12 and less than or equal 99.

String Length Validators

- These validators determine if a field's character length is greater than or equal or less than or equal a specified value
- Note that null value is considered an error
- You provide the minimum or maximum values as arguments in the field definition

validation.xml

```
<field property="password"
      depends="required,minLength">
  <arg0 key="password.display" />
  <arg1 name="minlength"
     key="${var:minlength}"
     resource="false"/>
  <var>
    <var-name>minlength</var-name>
    <var-value>6</var-value>
  </var>
</field>
```

ApplicationResources.properties

```
errors.minlength=The {0} must be at least  {1} characters
password.display=password
```

These validators let you ensure that the user enters strings that meet your minimum and maximum length requirements.

Here we show checking that the user enters a password of at least six characters.

Regular Expression Validators

- These validators use a determine if a field's contents match the specified regular expression
- You supply the regular expression as an argument in the field definition

validation.xml

```
<field property="ssn"
      depends="required,mask">
  <arg0 key="ssn.display" />
  <var>
    <var-name>mask</var-name>
    <var-value>
      [0-9]{3}-[0-9]{2}-[0-9]{4}
    </var-value>
  </var>
</field>
```

ApplicationResources.properties

```
errors.invalid=The {0} is not correctly formatted
ssn.display=Social Security number
```

Regular expressions are a powerful pattern-matching language that let you construct elaborate and exact specifications for the format of strings. We will not cover the regex language in detail here – consult any good textbook on the subject.

Here we supply a simple regex that specifies that a Social Security number should have three digits, a dash, two digits, another dash and then four digits.

Miscellaneous Validators

Validator	Base String	Variable(s)
email	errors.email	arg0 - property name
date	errors.date	arg0 - property name
creditcard	errors.creditcard	arg0 - property name

Struts provides three more standard validators:

The "date" validator checks to see if an entered string "looks" like a date. You can specify a "datePattern" attribute in validate.xml that defines the format of the date, but if you don't, the validator uses the DateFormat.SHORT format for the current locale. For more information on date formats, see the JavaDoc for the java.text.DateFormat and java.text.SimpleDateFormat classes.

The "email" validator does a simple check to see if an entered string "looks" like an email address (i.e. it has an @ and a dot). It does not actually verify that the email address works.

The "creditcard" validator uses a well-known algorithm (similar to a hash code) to determine if an entered string has valid digits for a credit-card number. It does not verify that the number actually corresponds to a valid credit card with an non-zero balance!

Enabling JavaScript in the Input Form

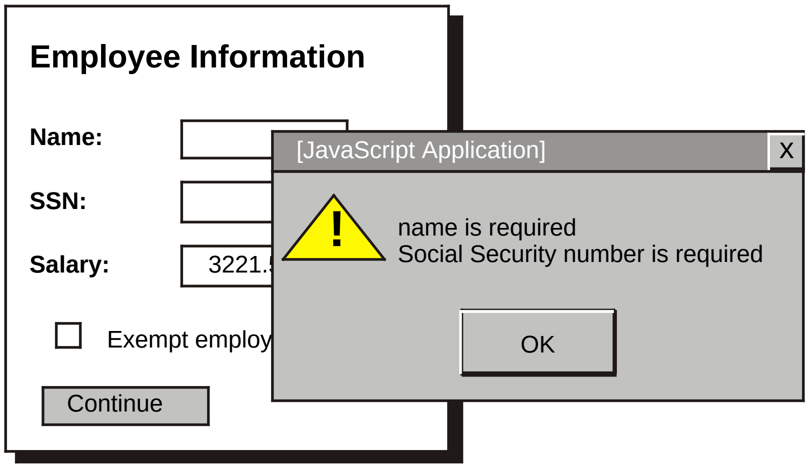
- All of the standard validators support client-side JavaScript
- To enable client-side validation, you must add some simple code to your input form

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

<html:javascript formName="MyForm" />

. . .

    <html:form action="My.action" method="post"
      onsubmit="return validateMyForm(this)">

. . .
```

To enable client-side validation, you can write a "javascript" element that specifies the name of the Struts ActionForm class for the input JSP and then, on the HTML form definition, add an "onsubmit" attribute with a value that looks like "return validateXXXX(this)", where the XXX is the name of the ActionForm class.

Client-Side Validation

- With client-side validation enabled, the emitted JavaScript pops up a message box if validation fails -- no roundtrip!
- The message box contains the same messages as would be generated by server-side validation



name is required
Social Security number is required

The generated JavaScript performs validation when the user clicks the submit button, but before the form is actually submitted to the server. If validation fails, the JavaScript code assembles the validation messages into a JavaScript "alert", which in most browsers, displays a message box of some sort.

It's important to emphasize that this all happens on the client, so no round trip to the server occurs. It's also important to remember that this checking only works if the browser supports JavaScript and if the user has scripting enabled in the browser.

Chapter Summary

In this chapter, you learned:

- The difference between client and server-side validation
- About the standard Struts validators