

Using the Struts Validator

Skill Level: Intermediate

[Brett McLaughlin](mailto:brett@newInstance.com) (brett@newInstance.com)

Author/editor

O'Reilly Media

16 Aug 2005

Follow along as Web development expert Brett McLaughlin guides you through the process of installing and configuring the Struts Validator component. The Validator, originally developed separately from and on top of Struts, is now an integral component of any professional Struts application programming. With the Validator, you can validate input in your Struts ActionForms. In this tutorial, you will learn to perform this validation declaratively, without touching your existing Java™ code.

Section 1. Before you start

About this tutorial

This tutorial guides you through the process of installing and configuring the Struts Validator component. I'll also show you how to use the Validator -- at least at a simple level -- and to ensure that it's working with your particular Struts configuration. And, if you've never used Struts (and are brave enough to keep working through this tutorial nonetheless), you'll want to review the [Appendix](#), which offers a crash course on Struts installation.

After completing this tutorial, you'll have the Validator set up and running, and you should be comfortable configuring Validator for use in your `ActionForms`. Additionally, you will have seen some very simple uses of the Validator, giving you a head start on your Struts/Validator application programming.

Who should take this tutorial?

This tutorial is written for Web developers who have some familiarity with Java technology, the Tomcat servlet engine, and the Struts application framework.

If you're using a servlet engine other than Tomcat, you will need to be comfortable with your servlet engine's setup and configuration; this tutorial assumes you're using Tomcat, and no extra detail is provided for non-Tomcat configurations. Check the Struts documentation for more information on installing Struts on servlet containers other than Tomcat (there's a link to the specific section of the documentation in [Resources](#)).

This tutorial deals specifically with configuring Struts, so you'll need to be at least passingly familiar with XML documents. You should also have administrative access to the machine you have Struts set up on; we'll be adding some JAR files and changing the core Struts setup to get the Validator up and running. You also might want to brush up on declarative programming (again, see [Resources](#) if declarative programming is new to you).

Prerequisites

To take this tutorial, you will need a machine -- or ISP -- that has a servlet engine, such as [Apache Tomcat](#), installed. I highly recommend that you run through this tutorial either locally, on a development machine, or on a non-production ISP account. In other words, don't try this out on a machine serving thousands of users, as you're going to have to make changes to your servlet container and you may have to restart that container several times.

Although neither is required for this tutorial, I'm using version 5.0 of the Java platform on Mac OS X. Some of the warnings and output captures you'll see in this tutorial reflect that. Additionally, I'm using Tomcat 5.5.9, which requires version 5.0 of the Java platform (unless you download a special bit of code that allows it to work with earlier JVMs). There is no special version of the Java platform or Tomcat required for this tutorial, although I strongly urge you to use at least version 5.0.

You also need a good text editor (or XML and Java editor, if you prefer), preferably one where you can have several windows open at once. There are a lot of files to configure, and using `vi` with a terminal window is going to get a little old. (That said, I'm using TextEdit on Mac OS X, so you don't need anything too fancy either.)

Finally, make sure you've downloaded the Struts engine and sample application (see the [Appendix](#)). I'll use the sample application to avoid going into too much detail about setting up Struts forms, which isn't the point of this tutorial.

There are a few more libraries that are required for the Validator itself to run, but we'll discuss these in [Installing the Validator framework](#).

Section 2. Building bulletproof applications with validation

The importance of validation

Great basketball teams are often lauded because they set solid picks, make great passes, and hit free throws. Great musicians play more than they need to and spend hours on just a note or two here or a certain phrasing there. Great automobiles have lots of extra features but run longer, have fewer engine problems, and are more reliable than their lesser competitors. All of these examples illustrate that great performers do "the little things." These little things may not be obvious, but they result in consistent, positive results.

When it comes to application programming, flashy user interfaces and impossibly complex threading won't make nearly as much of an impact on users as ensuring that their data is accurate. There's nothing worse than accidentally entering an invalid e-mail address or phone number into a Web form and having an application happily chug along, with no one the wiser. One of the little things that can really help in such a situation is *validation*.

Validation, at its simplest, is making sure that data is valid. That sounds a little silly, but *valid* can take on a variety of meanings, and that's where validation's power comes into play. A valid e-mail address might simply have at least one @ sign and one period (.) after the @ sign. However, a more complicated application might additionally require that an e-mail address be a certain number of characters long and be from one of a few specific domains. In either case, your code has to ensure that the data entered is acceptable; otherwise, you create a variety of possible error conditions:

- You may pass invalid data to another method, which has to then perform additional error checking, slowing your application down.
- You may pass invalid data to a method that *doesn't* perform error checking and thus cause your application to crash.
- You may insert invalid data into a database, which can result in an error the next time that data is read (which could be five minutes -- or five days

-- later).

In all of these cases, you're going to cause your users grief and give them a great excuse to go to a competitor's Web site or product.

The subject of this tutorial is validation. Granted, it's not an exciting, sexy topic, but it's one of those "little things" that makes a good application great.

Starting out with server-side validation

In a server-side programming environment like Struts, the easiest (and, to some degree, the most natural) approach to validation is to put several checks into your Java code. Specifically, the Struts `Action` that receives the data from a form seems like a good place to put validation code. You can grab values from the servlet request and check an e-mail or phone number's format, a first name's length, or a zip code's legitimacy. Then, if there are problems, you can show the submission form to the user again, presumably with error messages indicating the problems that occurred.

While this might sound good now, performing server-side validation is one of the worst means of achieving a good user experience. When users click **Submit**, **Save**, or **Enter** on a Web form and the page flashes (indicating that something is happening on the server), they don't expect to get their original submission back, complete with error messages. This problem is compounded if it takes a long time to process a request: your user has to wait, and then the application returns a set of error messages. The Web is a fast-paced medium, and users expect minimal lag, responsive applications, and dynamic user interfaces.

Instead of processing errors on the server, you should try to move validation to the client. Client-side validation can prevent the user from submitting a form containing erroneous data. It's also faster because there is no communication delay. You'll rarely find a good application these days without at least some client-side validation.

All that said, there are times when server-side validation -- or at least some form of server-side validation -- does make sense. Sometimes validation consists of checking formatting, length, and other trivial considerations (think about phone numbers, zip codes, state codes, and other basic data); other times it involves more complex calculations. You may need to ensure that a particular book is available for shipping to a certain zip code, for example. In these cases, you're probably going to have to allow a server request, as databases and other complex processing may be needed to ensure a proper response. However, users understand (in general) that these are more complex requests and will generally be patient for a response. In fact, this sort of computation is rarely seen as validation but is lumped in with the general business that your application serves. Still, for all but these most complex cases, you should move your validation to the client whenever possible.

Moving to JavaScript for validation

The most obvious choice for client-side validation is JavaScript. This popular scripting language is easy to learn, flexible, and available on every modern Web browser in existence (albeit in somewhat varying degrees). For many applications, JavaScript is as good as it gets. However, there are still some pitfalls.

First, JavaScript is still code, and you're going to have to write, test, debug, and deploy it just like any other set of code. If you move from 5-digit zip codes to 9-digit zip codes, your validation code will change, and you'll need to test out your changes on all the pages that they affect. This is time-intensive, although worthwhile for the snappy and user-pleasing results.

When using JavaScript, you can also be pretty sloppy -- you can easily drop little scriptlets into your HTML pages and create a maintenance nightmare. While it's not too difficult to exercise restraint, there is a temptation to just make "one quick fix" with a script inside a page and then never remove that script and rewrite the code in a JavaScript library.

Finally, you're probably going to have to carry around your JavaScript libraries from project to project. That's also not a huge hassle, but it's a consideration -- you'll be responsible for testing that code on every platform that you deploy to.

Don't get me wrong -- JavaScript is a great solution for validation, but if you're using Struts, there remains a third -- and even better -- option.

Going full tilt with the Struts Validator

Struts offers a great component called the *Validator*. The Validator plugs right into your Struts applications (something I'll cover in [Installing the Validator framework](#)), and even comes bundled with the latest Struts releases. You just drop in a few JAR files, and you're all set. But what's so great about the Validator? And why would you use it instead of JavaScript?

Well, you should realize that the Validator uses JavaScript for most of its execution. So you're not moving away from JavaScript, and you get the client-side validation that JavaScript excels at. However, the Validator removes many of the problems that JavaScript introduces. First, it's coded, tested, and debugged by thousands of Struts developers and users, reducing the amount of testing you are responsible for. (I'm definitely *not* implying that you don't have to test; the Validator reduces the testing load but doesn't remove it altogether.) Additionally, the Validator provides tons of common validation functions out of the box. So you don't have to code validators for e-mail addresses, phone numbers, zip codes, and other common pieces of data. How great is that?

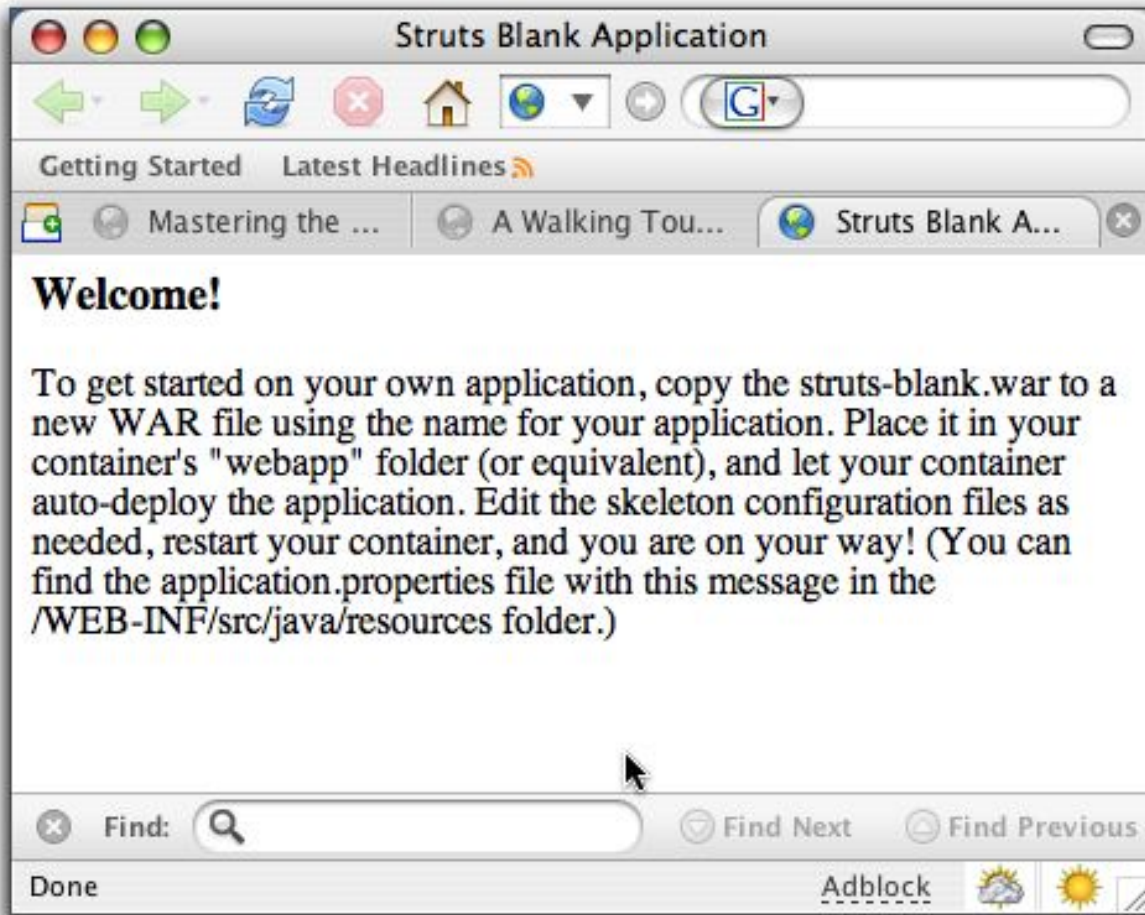
Perhaps most importantly, the Struts Validator works largely through configuration files, rather than inline HTML code. Through simple XML files, you can indicate what fields should be validated, and what types they should be validated against. Struts and the Validator take care of turning this into working JavaScript code, without any extra work on your part! While you may have to occasionally add new validation functions for application-specific data, the work of using these functions in your HTML is handled by Struts -- not a manual process. This is where the Validator truly excels and becomes worth its weight in gold. Convinced yet? Let's get to it then.

Section 3. Setting up a sample application

Starting with the struts-blank WAR

Before I can show you how to use the Validator, you need a sample application to work with. If you've installed Struts from the binary release (see the [Appendix](#) for details), then you've already got a great head start. Make sure that Tomcat is running, and navigate to <http://localhost:8080/struts-blank>. You should see something like Figure 1:

Figure 1. The struts-blank application provides a template for new applications



The text on this page tells you exactly what you need to do:

1. Locate the file `struts-blank.war` on your disk. (On my machine, it was at `/usr/local/jakarta-tomcat-5.5.9/webapps/struts-blank.war`.)
2. Copy this file to a location where you can work with it, whether it's in a development directory or on your desktop.
3. Rename the file `test-validation.war`. This will be the basis of the application you'll develop in this article to test validation out.

Now you need to expand the WAR file into a set of directories that you can work with conveniently. Create a directory to work within; I usually name it *staging* or something similar. Once you've created that directory, use the `jar` command with the `xvf` option to expand this file into your new directory, as shown in Listing 1:

Listing 1. WAR file directories

```
[bmclaugh:/usr/local/java]$ ls
jakarta-struts-1.2.4      src                    xalan-j_2_6_0
jakarta-tomcat-5.5.9    test-validation.war
[bmclaugh:/usr/local/java]$ mkdir staging
[bmclaugh:/usr/local/java]$ cd staging
[bmclaugh:/usr/local/java/staging]$ jar xvf ../test-validation.war
  created: META-INF/
  inflated: META-INF/MANIFEST.MF
  created: WEB-INF/
  created: WEB-INF/classes/
  created: WEB-INF/classes/resources/
  created: WEB-INF/lib/
  created: WEB-INF/src/
  created: WEB-INF/src/java/
  created: WEB-INF/src/java/resources/
  created: pages/
  inflated: WEB-INF/classes/MessageResources.properties
  inflated: WEB-INF/classes/resources/MessageResources.properties
  inflated: WEB-INF/lib/commons-beanutils.jar
  inflated: WEB-INF/lib/commons-collections.jar
  inflated: WEB-INF/lib/commons-digester.jar
  inflated: WEB-INF/lib/commons-fileupload.jar
  inflated: WEB-INF/lib/commons-logging.jar
  inflated: WEB-INF/lib/commons-validator.jar
  inflated: WEB-INF/lib/jakarta-oro.jar
  inflated: WEB-INF/lib/struts.jar
  inflated: WEB-INF/src/README.txt
  inflated: WEB-INF/src/build.xml
  inflated: WEB-INF/src/java/resources/application.properties
  inflated: WEB-INF/struts-bean.tld
  inflated: WEB-INF/struts-config.xml
  inflated: WEB-INF/struts-html.tld
  inflated: WEB-INF/struts-logic.tld
  inflated: WEB-INF/struts-nested.tld
  inflated: WEB-INF/struts-tiles.tld
  inflated: WEB-INF/tiles-defs.xml
  inflated: WEB-INF/validation.xml
  inflated: WEB-INF/validator-rules.xml
  inflated: WEB-INF/web.xml
  inflated: index.jsp
  inflated: pages/Welcome.jsp
```

You can make changes to these files and customize the application to your liking. What's so nice about this approach is that all the setup files are in place, and you don't have to worry about getting all the right WAR files downloaded. What developer doesn't like a nice handy shortcut?

Changing the Welcome page

First, you need to add a link on the main page; this link will take users to a simple form where some validation rules can be demonstrated. If you access the sample application, you can quickly see that the default page shown is `Welcome.jsp`, stored in the `pages` directory of your application structure. Open up that file in a text editor or IDE.

The version provided by the `struts-blank` WAR is a good start; to change the text

shown, you don't even have to touch this file, as all the text is stored in a MessageResources file (which I'll talk about in a moment). All you need to do, then, is add a simple link. Add the bold lines shown in Listing 2 to your Welcome.jsp file:

Listing 2. Welcome.jsp file

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>

<html:html locale="true">
<head>
<title><bean:message key="welcome.title"/></title>
<html:base/>
</head>
<body bgcolor="white">

<logic:notPresent name="org.apache.struts.action.MESSAGE" scope="application">
  <font color="red">
    ERROR: Application resources not loaded -- check servlet container
    logs for error messages.
  </font>
</logic:notPresent>

<h3><bean:message key="welcome.heading"/></h3>
<p><bean:message key="welcome.message"/></p>
<p>
<ul>
<li><html:link action="/TestSimpleValidation">
  <bean:message key="welcome.test-validation" />
</html:link></li>
</ul>
</p>

</body>
</html:html>
```

The `html:link` tag is a pretty basic construct in Struts. Here, it's used to link to a Struts Action that will load the page with some validation examples on it. Before getting to that, though, there are some properties that need to be changed (`welcome.title`, `welcome.heading`, and `welcome.message`) and one that needs to be added (`welcome.test-validation`).

These properties are in the MessageResources.properties file, located in the WEB-INF/classes directory of your application's structure. Open that file up, and go to the bottom of the file. Make the changes shown here in bold:

```
# -- welcome --
welcome.title=Struts Validator Test Application
welcome.heading=Welcome to the Validation Tester Application!
welcome.message=This is a simple application meant to test and
demonstrate the Struts Validator component.
welcome.test-validation=Test out some simple uses of the Struts Validator
```

Now you just need to add a simple `forward` into your `struts-config.xml` file (this file should be in the `WEB-INF` folder of your application). Look for the `action-mappings` element and add this in:

```
<action-mappings>
  <action path="/TestSimpleValidation"
    forward="/pages/test-validation.jsp" />

  <!-- Other Action mappings -->
</action-mappings>
```

With these changes in place, you've got an initial Welcome page ready to go. On the next panel, you will test this out by redeploying the application.

Redeploying your application

Any time you want to redeploy, there are just a few simple steps to follow:

1. Use the `jar` command to compact your application back into a WAR file, shown in Listing 3:

Listing 3. The `jar` command

```
[bmclaugh:/usr/local/java]$ cd staging
[bmclaugh:/usr/local/java/staging]$ jar cvf ../test-validation.war *
added manifest
ignoring entry META-INF/
ignoring entry META-INF/MANIFEST.MF
adding: WEB-INF/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/MessageResources.properties(in = 1167)
(out= 488)(deflated 58%)
adding: WEB-INF/classes/resources/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/resources/MessageResources.properties(in = 1480)
(out= 655)(deflated 55%)
adding: WEB-INF/lib/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/lib/commons-beanutils.jar(in = 118726) (out= 105729)
(deflated 10%)
adding: WEB-INF/lib/commons-collections.jar(in = 175426) (out= 144506)
(deflated 17%)
adding: WEB-INF/lib/commons-digester.jar(in = 109096) (out= 99033)
(deflated 9%)
adding: WEB-INF/lib/commons-fileupload.jar(in = 22379) (out= 19246)
(deflated 13%)
adding: WEB-INF/lib/commons-logging.jar(in = 38015) (out= 34595)
(deflated 8%)
adding: WEB-INF/lib/commons-validator.jar(in = 84260) (out= 76342)
(deflated 9%)
adding: WEB-INF/lib/jakarta-oro.jar(in = 65261) (out= 56142)
(deflated 13%)
adding: WEB-INF/lib/struts.jar(in = 526578) (out= 480962)
(deflated 8%)
adding: WEB-INF/src/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/src/build.xml(in = 3672) (out= 1080)(deflated 70%)
adding: WEB-INF/src/java/(in = 0) (out= 0)(stored 0%)
```

```

adding: WEB-INF/src/java/resources/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/src/java/resources/application.properties(in = 1480)
(out= 655)(deflated 55%)
adding: WEB-INF/src/README.txt(in = 1923) (out= 837)(deflated 56%)
adding: WEB-INF/struts-bean.tld(in = 8860) (out= 771)(deflated 91%)
adding: WEB-INF/struts-config.xml(in = 6665) (out= 1994)(deflated 70%)
adding: WEB-INF/struts-html.tld(in = 67559) (out= 2069)(deflated 96%)
adding: WEB-INF/struts-logic.tld(in = 14731) (out= 830)(deflated 94%)
adding: WEB-INF/struts-nested.tld(in = 65059) (out= 2086)(deflated 96%)
adding: WEB-INF/struts-tiles.tld(in = 7842) (out= 717)(deflated 90%)
adding: WEB-INF/tiles-defs.xml(in = 1379) (out= 535)(deflated 61%)
adding: WEB-INF/validation.xml(in = 2121) (out= 550)(deflated 74%)
adding: WEB-INF/validator-rules.xml(in = 12254) (out= 1628)(deflated 86%)
adding: WEB-INF/web.xml(in = 1942) (out= 582)(deflated 70%)
adding: index.jsp(in = 276) (out= 188)(deflated 31%)
adding: pages/(in = 0) (out= 0)(stored 0%)
adding: pages/Welcome.jsp(in = 657) (out= 318)(deflated 51%)

```

2. You can ensure that your WAR file contains what it should with the `tvf` option on `jar`:

Listing 4. The `tvf` option

```

[bmclaugh:/usr/local/java/staging]$ cd ..
[bmclaugh:/usr/local/java]$ jar tvf test-validation.war
  0 Thu May 19 17:13:34 CDT 2005 META-INF/
 70 Thu May 19 17:13:34 CDT 2005 META-INF/MANIFEST.MF
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/
  0 Thu May 19 16:55:00 CDT 2005 WEB-INF/classes/
1167 Thu May 19 16:55:00 CDT 2005 WEB-INF/classes/MessageResources.properties
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/classes/resources/
1480 Thu May 19 16:51:20 CDT 2005 WEB-INF/classes/resources/
MessageResources.properties
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/
118726 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-beanutils.jar
175426 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-collections.jar
109096 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-digester.jar
22379 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-fileupload.jar
38015 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-logging.jar
84260 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/commons-validator.jar
65261 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/jakarta-oro.jar
526578 Thu May 19 16:51:20 CDT 2005 WEB-INF/lib/struts.jar
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/
3672 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/build.xml
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/java/
  0 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/java/resources/
1480 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/java/resources/application.properties
1923 Thu May 19 16:51:20 CDT 2005 WEB-INF/src/README.txt
8860 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-bean.tld
6665 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-config.xml
67559 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-html.tld
14731 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-logic.tld
65059 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-nested.tld
7842 Thu May 19 16:51:20 CDT 2005 WEB-INF/struts-tiles.tld
1379 Thu May 19 16:51:20 CDT 2005 WEB-INF/tiles-defs.xml
2121 Thu May 19 16:51:20 CDT 2005 WEB-INF/validation.xml
12254 Thu May 19 16:51:20 CDT 2005 WEB-INF/validator-rules.xml
1942 Thu May 19 16:51:20 CDT 2005 WEB-INF/web.xml
276 Thu May 19 16:51:20 CDT 2005 index.jsp
  0 Thu May 19 16:53:24 CDT 2005 pages/
657 Thu May 19 16:51:20 CDT 2005 pages/Welcome.jsp

```

3. Copy your new WAR file to your Tomcat webapps directory:

```
[bmclaugh:/usr/local/java]$ cp test-validation.war jakarta-tomcat-5.5.9/webapps/
```

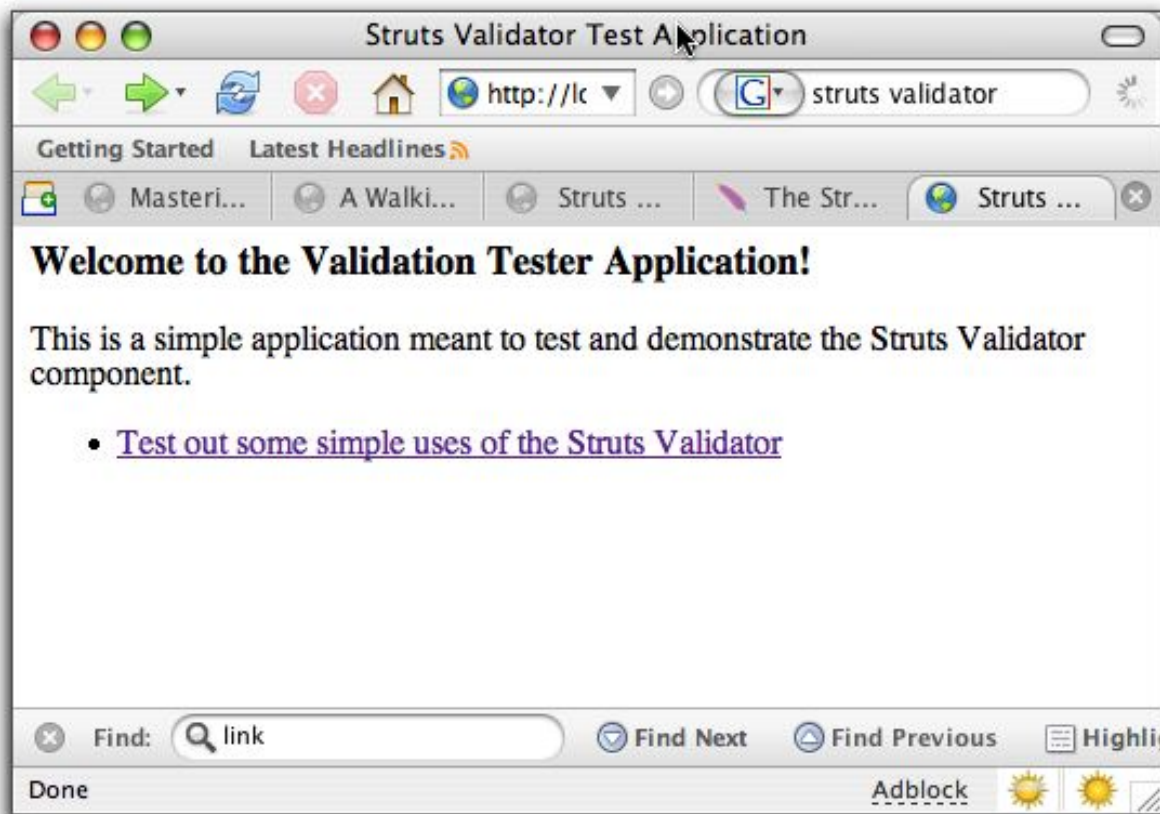
4. Access your application at <http://localhost:8080/test-validation/>.

You can write scripts or Ant files to perform these tasks for you, or you can perform them by hand. In any case, it's a nice quick way to redeploy your application as you change it (and you'll be changing it a lot in this tutorial).

Checking the Welcome page

With your application redeployed, check out <http://localhost:8080/test-validation/>. You should see a page that looks like Figure 2:

Figure 2. The changes to `Welcome.jsp` and message resources creates a new Welcome screen



At this point, there's no meat to the application; clicking on the new link gives you a nasty error. But that's OK -- you've got a good beginning, and you're now ready to

deal with the Validator directly.

Section 4. Installing the Validator framework

Required libraries

The Struts Validator is a pretty complicated piece of software, and it depends on several other libraries to function properly. Just as you need several JAR files to run Struts in a servlet engine, you need several more for the Validator to work. Most importantly, you need the [Jakarta ORO](#) package, which handles regular expressions.

The Validator also uses the [Jakarta Commons BeanUtils](#), [Jakarta Commons Logging](#), [Jakarta Commons Collections](#), and [Jakarta Commons Digester](#) packages. You'll need the JAR files for each of these either in your Tomcat common/lib directory or your Web application's WEB-INF/lib directory.

Finally, the Struts Validator is built on (yet another) Jakarta Commons package: the [Jakarta Commons Validator](#). So there's yet another JAR file you'll need available. That's a lot of JAR files, and there's still more to deal with. But before you start downloading, keep reading. Once I show you all the requirements, I'll also show you a shortcut to getting Validator support in your applications without all this downloading and manual configuration.

Validation rules

Once you've got the Validator libraries in place, you'll need two XML files: validation-rules.xml and validator.xml. The validation-rules.xml file is more or less static, so I'll deal with it first. This file specifies the validation rules available; because the Validator comes with several default rules, you will just want to locate a working copy of this file and copy it into your own application's WEB-INF directory.

This tends to be a pretty long file, so I'm going to include just a small fragment of it in Listing 5 to show you how it looks:

Listing 5. The validation-rules.xml file

```
<DOCTYPE form-validation PUBLIC
    "-//Apache Software Foundation//DTD Commons
    Validator Rules Configuration 1.1.3//EN"
```

```

    "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
<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.ActionMessages,
                    javax.servlet.http.HttpServletRequest "
      msg="errors.required"/>

    <validator name="requiredif"
      classname="org.apache.struts.validator.FieldChecks"
      method="validateRequiredIf"
      methodParams=" java.lang.Object,
                    org.apache.commons.validator.ValidatorAction,
                    org.apache.commons.validator.Field,
                    org.apache.struts.action.ActionMessages,
                    org.apache.commons.validator.Validator,
                    javax.servlet.http.HttpServletRequest "
      msg="errors.required"/>

    <validator name="validwhen"
      msg="errors.required"
      classname="org.apache.struts.validator.validwhen.ValidWhen"
      method="validateValidWhen"
      methodParams=" java.lang.Object,
                    org.apache.commons.validator.ValidatorAction,
                    org.apache.commons.validator.Field,
                    org.apache.struts.action.ActionMessages,
                    org.apache.commons.validator.Validator,
                    javax.servlet.http.HttpServletRequest "/>

    <validator name="minlength"
      classname="org.apache.struts.validator.FieldChecks"
      method="validateMinLength"
      methodParams=" java.lang.Object,
                    org.apache.commons.validator.ValidatorAction,
                    org.apache.commons.validator.Field,
                    org.apache.struts.action.ActionMessages,
                    javax.servlet.http.HttpServletRequest "
      depends=" "
      msg="errors.minlength"
      jsFunction="org.apache.commons.validator.javascript.validateMinLength"/>

    <!--
      This simply allows struts to include the validateUtilities into a page, it should
      not be used as a validation rule.
    -->
    &lt;validator name="includeJavaScriptUtilities"
      classname=" "
      method=" "
      methodParams=" "
      depends=" "
      msg=" "
      jsFunction="org.apache.commons.validator.javascript.validateUtilities"/>

  </global>
</form-validation>

```

Don't worry too much about the intricacies of this file; unless you're writing your own validation rules, you can simply accept it as is and use this default version.

Adding error messages to your application

Most versions of validation-rules.xml contain something like that in Listing 6 in a comment at the head of the file:

Listing 6. The validation-rules.xml comment

```
These are the default error messages associated with
each validator defined in this file. They should be
added to your projects ApplicationResources.properties
file or you can associate new ones by modifying the
pluggable validators msg attributes in this file.

# Struts Validator Error Messages
errors.required={0} is required.
errors.minlength={0} can not be less than {1} characters.
errors.maxlength={0} can not be greater than {1} characters.
errors.invalid={0} is invalid.

errors.byte={0} must be a byte.
errors.short={0} must be a short.
errors.integer={0} must be an integer.
errors.long={0} must be a long.
errors.float={0} must be a float.
errors.double={0} must be a double.

errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is an invalid credit card number.
errors.email={0} is an invalid e-mail address.
```

The message at the top of this comment is good advice. The Validator will use these properties -- whether you follow these instructions or not -- when it outputs error messages. If you don't insert these into your application's resources, you'll end up with blank error messages, which of course look like no error messages at all. There are few things more frustrating to a user than trying to submit a form, having that form rejected, and then not getting any sort of feedback as to what went wrong.

These properties are best located in a file like MessageResources.properties, commonly located in your application's WEB-INF/classes directory:

Listing 7. MessageResources.properties

```
<b># -- standard errors --
errors.header=<UL>
errors.prefix=<LI>
errors.suffix=</LI>
errors.footer=</UL>
# -- validator --
errors.invalid={0} is invalid.
errors.maxlength={0} can not be greater than {1} characters.
errors.minlength={0} can not be less than {1} characters.
errors.range={0} is not in the range {1} through {2}.
errors.required={0} is required.
```

```

errors.byte={0} must be an byte.
errors.date={0} is not a date.
errors.double={0} must be an double.
errors.float={0} must be an float.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.short={0} must be an short.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
# -- other --
errors.cancel=Operation cancelled.
errors.detail={0}
errors.general=The process did not complete. Details should follow.
errors.token=Request could not be completed. Operation is not in sequence.</b>
# -- welcome --
welcome.title=Struts Validator Test Application
welcome.heading>Welcome to the Validation Tester Application!
welcome.message=This is a simple application meant to test and
    demonstrate the Struts Validator component.
welcome.test-validation=Test out some
    simple uses of the Struts Validator

```

Again, though, hold off on making these changes, as there's an easier way (those of you who have already cracked open the WEB-INF/classes/MessageResources.properties file may already have an idea about this shortcut). If you want to use different error messages, you can make changes here. For example, if you wanted to make the error for an invalid e-mail address a little nicer, you might change it to something like this:

```
errors.email=You have entered an invalid e-mail address ({0}). Please try again.
```

This is a great way to customize your error messages, all without touching an actual line of code (and therefore avoiding recompilation, redeployment, and lots of retesting).

Connecting the Validator to your application

Now you need to let your Struts application know about the Validator. Here, I'm referring to the component as a whole, and not specific uses of the Validator (something I'll get to in [Using the Validator in an application](#)). You'll need to use Struts' `plugin` element, the method for letting Struts know about components that it should integrate into an application. You'll need the following entry in your `struts-config.xml` file, located in your application's WEB-INF directory:

```

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

```

The `className` attribute tells Struts what class to load; that class should implement Struts' `PlugIn` interface, as `ValidatorPlugIn` does. Then, you can have any number of plugin-specific `set-property` entries; for the Validator, you only need one. This should set the property of `pathnames` to the value of the paths to your `validator-rules.xml` and `validation.xml` files. If you want to store these somewhere else, you can specify that alternate location here, in the `plugin` element.

With these steps complete, all you need to do is write a `validation.xml` file specific to your application's forms, actions, and validation needs. But first, let me give you that shortcut to avoid all this tedious setup for the Validator.

The value of the struts-blank application

If you followed my instructions earlier and copied the `struts-blank.war` file to a new file -- using it as your base for developing Struts applications -- then you've uncovered a huge secret: `struts-blank.war` comes preconfigured to use the Struts Validator! Take a look at your sample application's `WEB-INF/lib` folder, and you'll see all the libraries the Validator needs:

- **commons-beanutils.jar**: Commons BeanUtils
- **commons-collections.jar**: Commons Collections
- **commons-digester.jar**: Commons Digester
- **commons-logging.jar**: Commons Logging
- **commons-validator.jar**: Commons Validator
- **jakarta-oro.jar**: Jakarta ORO

If you look in `WEB-INF/classes/MessageResources.properties`, all of the Validator `error` properties are defined. In `WEB-INF`, you'll see a default version of `validation-rules.xml`, and it contains all of the Validator default validation rules, ready to roll. There's a very basic version of `validation.xml`, ready for you to modify (the subject of the next section of this tutorial). And, best of all, `struts-config.xml` has the Validator `plugin` element set up.

This is the real beauty of `struts-blank.war`, at least in my opinion: I never have to remember what JAR files and configuration steps are needed to get the Validator working. I simply copy that file to a new location, rename it to my application's name, and start working. Even if I have to make a few changes to existing classes or files (like adding a link to the Welcome page, or removing some JSPs), it's worth it to me to not have to worry about Validator setup.

You're certainly welcome to perform the steps outlined in this section every time you develop a new Struts application, but I'd rather copy and rename `struts-blank.war`

and get to work.

Section 5. Using the Validator in an application

Creating a form to test validation

When we left the sample application, you had a Welcome page and a link to another JSP, pages/test-validation.jsp. You're now ready to put that page in place. To start with, this will be just a plain old JSP with a basic form; once you get your simple Struts application running, I'll show you how to come back in and add the validation logic. For now, start with the basic JSP shown in Listing 8:

Listing 8. Basic JSP

```
<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>

<html:xhtml />
<html>
  <head>
    <title><bean:message key="valid.title" /></title>
  </head>

  <html:form action="/SubmitValid" focus="username">
    <table border="0" width="100%">
      <tr>
        <th align="right">
          <bean:message key="prompt.username" />:
        </th>
        <td align="left">
          <html:text property="username" size="10"
maxlength="10" />
        </td>
      </tr>
      <tr>
        <th align="right">
          <bean:message key="prompt.password" />:
        </th>
        <td align="left">
          <html:password property="password" size="16"
maxlength="16"
                      redisplay="false" />
        </td>
      </tr>
      <tr>
        <th align="right">
          <bean:message key="prompt.phone" />:
        </th>
        <td align="left">
          <html:text property="phone" size="14" maxlength="14"
/ >
        </td>
```

```

    </tr>
    <tr>
      <th align="right">
        <bean:message key="prompt.email" />:
      </th>
      <td align="left">
        <html:text property="email" size="20"
maxlength="100" />
      </td>
    </tr>
    <tr>
      <th align="right">
        <bean:message key="prompt.url" />:
      </th>
      <td align="left">
        <html:text property="url" size="20" maxlength="100"
/>
      </td>
    </tr>
    <tr>
      <td align="right">
        <html:reset />
      </td>
      <td align="left">
        <html:submit property="Submit" value="Submit" />
      </td>
    </tr>
  </table>
</html:form>
</body>
</html>

```

There's little to add to the listing. As you can see, it creates a form and then provides plenty of opportunity to enter invalid data. Notice the extensive use of properties for label names; you should add those into your `WEB-INF/classes/MessageResources.properties` file now. Place them at the bottom of the file, after your existing entries:

```

# -- validation test page --
valid.title=Simple Validation Test Form
prompt.username=Username
prompt.password=Password
prompt.phone=Phone Number
prompt.email=E-Mail Address
prompt.url=URL (Website Address)

```

Note: It may seem like overkill to use properties on such a simple JSP page in a tutorial. However, this is simply good coding practice. You can localize these properties, change them easily, and reuse them, and it takes almost no additional development time. Get used to taking advantage of best practices like this, even in your sample applications and prototyping. It will really pay off in the long run.

Trying to visit this page will still generate an error, though; there's still plenty of work to do (sometimes the things that make Struts nice -- declarative exceptions, highly configurable forms, and the like -- make it hard to get up and running quickly). At this stage, you should get a nasty message that looks like Listing 9:

Listing 9. Error message

```

javax.servlet.ServletException: Cannot retrieve mapping for action /SubmitValid
  org.apache.jasper.runtime.PageContextImpl.doHandlePageException(
    PageContextImpl.java:848)
  org.apache.jasper.runtime.PageContextImpl.handlePageException(
    PageContextImpl.java:781)
  org.apache.jsp.pages.test_002dvalidation_jsp._jspService(
    org.apache.jsp.pages.test_002dvalidation_jsp:102)
  org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  org.apache.jasper.servlet.JspServletWrapper.service(
    JspServletWrapper.java:322)
  org.apache.jasper.servlet.JspServlet.serviceJspFile(
    JspServlet.java:291)
  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:241)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  org.apache.struts.action.RequestProcessor.doForward(
    RequestProcessor.java:1056)
  org.apache.struts.tiles.TilesRequestProcessor.doForward(
    TilesRequestProcessor.java:261)
  org.apache.struts.action.RequestProcessor.internalModuleRelativeForward(
    RequestProcessor.java:994)
  org.apache.struts.tiles.TilesRequestProcessor.internalModuleRelativeForward(
    TilesRequestProcessor.java:343)
  org.apache.struts.action.RequestProcessor.processForward(
    RequestProcessor.java:553)
  org.apache.struts.action.RequestProcessor.process(
    RequestProcessor.java:211)
  org.apache.struts.action.ActionServlet.process(
    ActionServlet.java:1164)
  org.apache.struts.action.ActionServlet.doGet(
    ActionServlet.java:397)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)

```

root cause

```

javax.servlet.jsp.JspException: Cannot retrieve mapping for action /SubmitValid
  org.apache.struts.taglib.html.FormTag.lookup(FormTag.java:723)
  org.apache.struts.taglib.html.FormTag.doStartTag(FormTag.java:419)
  org.apache.jsp.pages.test_002dvalidation_jsp._jspx_meth_html_form_0(
    org.apache.jsp.pages.test_002dvalidation_jsp:150)
  org.apache.jsp.pages.test_002dvalidation_jsp._jspService(
    org.apache.jsp.pages.test_002dvalidation_jsp:92)
  org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  org.apache.jasper.servlet.JspServletWrapper.service(
    JspServletWrapper.java:322)
  org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:291)
  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:241)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  org.apache.struts.action.RequestProcessor.doForward(
    RequestProcessor.java:1056)
  org.apache.struts.tiles.TilesRequestProcessor.doForward(
    TilesRequestProcessor.java:261)
  org.apache.struts.action.RequestProcessor.internalModuleRelativeForward(
    RequestProcessor.java:994)
  org.apache.struts.tiles.TilesRequestProcessor.internalModuleRelativeForward(
    TilesRequestProcessor.java:343)
  org.apache.struts.action.RequestProcessor.processForward(
    RequestProcessor.java:553)
  org.apache.struts.action.RequestProcessor.process(
    RequestProcessor.java:211)
  org.apache.struts.action.ActionServlet.process(ActionServlet.java:1164)
  org.apache.struts.action.ActionServlet.doGet(ActionServlet.java:397)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:802)

```

You Struts veterans will recognize what this error message means: You've defined an action in `pages/validation-test.jsp` called `SubmitValid`, but there is no matching `action` element in the `struts-config.xml` file. There are also some other related problems: You need a `form-bean` as well. I'll deal with both of these problems next.

Configuring the validation test page

First, let's handle the missing `SubmitValid` action. Add the entry in Listing 10 to your `struts-config.xml` file:

Listing 10. Entry for `struts-config.xml` file

```
<action-mappings>
  <!-- Default "Welcome" action -->
  <!-- Forwards to Welcome.jsp -->
  <action
    path="/Welcome"
    forward="/pages/Welcome.jsp" />

  <action path="/TestSimpleValidation"
    forward="/pages/test-validation.jsp" />
    <action path="/SubmitValid"
      type="com.ibm.struts.validation.ValidationAction"
      name="ValidationForm"
      scope="request"
      validate="true"
      input="/pages/test-validation.jsp">
      <forward name="success" path="/pages/success.jsp" redirect="true"/>
      <forward name="failure" path="/pages/test-validation.jsp"
        redirect="true" />
    </action>
  </action>

  <!-- Other action elements -->
</action-mappings>
```

This doesn't do a lot; it simply associates the target of `test-validation.jsp`'s form to a new class, `com.ibm.struts.validation.ValidationAction`. This isn't a class I've shown you yet; you'll code it up shortly. The rest should look fairly typical to any Struts developer. It indicates the name of the form to pass to the action, the scope, and the input page. It also turns on validation, for fairly obvious reasons.

Now, when you access the page, you'll get a new error (assuming you made the changes above, and redeployed your application):

Listing 11. New error message

```
javax.servlet.ServletException:
  Cannot retrieve definition for form bean ValidationForm on action /SubmitValid
  org.apache.jasper.runtime.PageContextImpl.doHandlePageException
  (PageContextImpl.java:848)
  org.apache.jasper.runtime.PageContextImpl.handlePageException(PageContextImpl.java:781)
  org.apache.jsp.pages.test_002dvalidation_jsp._jspService
```

```

(org.apache.jsp.pages.test_002dvalidation_jsp:102)
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:322)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:291)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:241)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
org.apache.struts.action.RequestProcessor.doForward(RequestProcessor.java:1056)
org.apache.struts.tiles.TilesRequestProcessor.doForward(TilesRequestProcessor.java:261)
org.apache.struts.action.RequestProcessor.internalModuleRelativeForward
(RequestProcessor.java:994)
org.apache.struts.tiles.TilesRequestProcessor.internalModuleRelativeForward
(TilesRequestProcessor.java:343)
org.apache.struts.action.RequestProcessor.processForward(RequestProcessor.java:553)
org.apache.struts.action.RequestProcessor.process(RequestProcessor.java:211)
org.apache.struts.action.ActionServlet.process(ActionServlet.java:1164)
org.apache.struts.action.ActionServlet.doGet(ActionServlet.java:397)
javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)

```

This is another easy problem to fix; you just need to add a new `form-bean` element -- also to `WEB-INF/struts.config.xml` -- that defines a JavaBean for your form, shown in Listing 12:

Listing 12. The form-bean element

```

<form-beans>
  <!-- Other form-bean listings -->

  <form-bean name="ValidationForm"
            type="org.apache.struts.action.DynaActionForm">
    <form-property name="username" type="java.lang.String" />
    <form-property name="password" type="java.lang.String" />
    <form-property name="phone" type="java.lang.String" />
    <form-property name="email" type="java.lang.String" />
    <form-property name="url" type="java.lang.String" />
  </form-bean>
</form-beans>

```

Now you've got a working configuration. Redeploy these changes, visit the Welcome page, and click on the link. You should see the validation test page, as shown in Figure 3:

Figure 3. Now the Welcome screen takes you to the test form

Simple Validation Test Form

2500 Desert Falls ▾ Freelance ▾ O'Reilly ▾

A Walking Tour of ... Using the Struts Va... Simple Validation ...

Username:

Password:

Phone Number:

E-Mail Address:

URL (Website Address):

Reset Submit

Find: Find Next Find Previous Highlight

Done Adblock

This is hardly the most beautiful form you'll ever see, but it serves our purposes just fine. You need to put an `Action` in place to process it, and then all that's left is some validation -- which is the fun part, of course!

Adding a custom action for processing

Because there isn't any business logic to worry about -- this is just a sample application -- writing a custom `Action` for processing the validation form is easy, as shown in Listing 13:

Listing 13. A custom Action

```
package com.ibm.struts.validation;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
```

```
public class ValidationAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                HttpServletRequest req,
                                HttpServletResponse res)
        throws Exception {
        ActionForward forward = null;

        // Perform some sort of business logic with the data. In this sample,
        // we don't care about this; in fact, if the application got here,
        // they've already passed validation, and we just need to return success

        forward = mapping.findForward("success");
        return forward;
    }
}
```

This, of course, is the action referenced in `struts-config.xml`'s `SubmitValid` action. In this case, it does next to nothing -- just passes control on to the Struts controller with a "success" forward. This matches up to the `struts-config.xml` file and requests the `/pages/success.jsp` JSP, shown here:

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>

<html:html locale="true">
<head>
<title>Welcome</title>
<html:base/>
</head>
<body bgcolor="white">

<h3>Validation Successful!</h3>

</body>
</html:html>
```

It's nothing fancy, of course, as this is a tutorial about validation, not page design.

What is worth noting, before you dismiss this as so much fluff, is that there has been no validation code yet! While you might find that disappointing, it's actually worth getting excited about. You haven't had to create a custom `ActionForm` and implement the `validate()` method yourself, and your `Action` doesn't know anything about validity -- in fact, it won't even be accessed if there are validation problems. This means that your code is completely clean of validation logic, and, as you'll see in the next section, it's going to stay that way.

In fact, this is one of the primary reasons I'm spending so much time walking through application setup. Your code, your business logic, and even most of your configuration is devoid of validation logic. You're not going to have to worry about redeploying your forms and actions every time you change the allowed length of a password. In fact, you'll only have to change a small XML file. But now I'm getting ahead of myself; in the next section, you'll (finally) begin to look at what you *do* have to do to take advantage of validation.

Section 6. Working with the Validator

Struts without the Validator

Before talking about what you do to use the Validator, let me (briefly) detail what you'd have to do to validate content *without* the Validator. Recall the `form-bean` element added to `WEB-INF/struts-config.xml` earlier:

```
<form-bean name="ValidationForm"
  type="org.apache.struts.action.DynaActionForm">
  <form-property name="username" type="java.lang.String" />
  <form-property name="password" type="java.lang.String" />
  <form-property name="phone" type="java.lang.String" />
  <form-property name="email" type="java.lang.String" />
  <form-property name="url" type="java.lang.String" />
</form-bean>
```

If you didn't have the Validator installed and available and wanted to perform server-side validation, you'd replace the type of the form -- currently set to `org.apache.struts.action.DynaActionForm` -- with a custom type, something like `com.ibm.struts.ValidationForm`. That class would presumably extend the default Struts form, `org.apache.struts.action.ActionForm`. Then, you'd implement the `validate()` method. It would probably look something like this:

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

    // Do all sorts of validation
    if ((getUsername() == null) || (getUsername().length() < 1)) {
        errors.add("username", new ActionMessage("validation.errors.username.required"));
    } else if (getUsername().length() < 8) {
        errors.add("username", new ActionMessage("validation.errors.username.too-short"));
    } // ... etc.

    return errors;
}
```

Not only do you now need to create a custom class for every form in your application, but you've also violated one of those principles mentioned back in the introduction to this tutorial -- your validation is server-based. Every request must go to the servlet engine, get delegated to Struts, get passed to the right `ActionForm`, be processed, and then returned (to Struts, then to the servlet container, then to the user). Nasty stuff, isn't it?

You could move this into JavaScript (the next best option), but that's a pain as well, for all the reasons already mentioned. Additionally, your JavaScript module (you are writing your scripts in a module file, and not directly into your JSPs, right?) can't access the Struts property files, meaning all those nice error messages you added to `MessageResources.properties` cannot be used in validation. So you're losing some of Struts' nicest features: modularity and easy internationalization.

Clearly, there must be a better way. And, thankfully, it's now time to get on to more Validator code.

Dynamic validation and the Validator

If you've closed your `struts-config.xml` file, open it up again. It's time to revisit that `form-bean` element one more time. You need to change the form type, but not to a custom class. Instead, use the Validator-provided class, `org.apache.struts.validator.DynaValidatorForm` shown here:

```
<form-bean name="ValidationForm"
  type="org.apache.struts.validator.DynaValidatorForm">
  <form-property name="username" type="java.lang.String" />
  <form-property name="password" type="java.lang.String" />
  <form-property name="phone" type="java.lang.String" />
  <form-property name="email" type="java.lang.String" />
  <form-property name="url" type="java.lang.String" />
</form-bean>
```

Make this change, save your modified configuration file, and redeploy your application. If you go to the validation form, enter some dummy values (or enter no values at all), and click **Submit**. You'll see ... well, the same `success.jsp` page you saw before. In fact, making this change hasn't notably altered the application. That's because you haven't specified any validation rules to apply. However, you've now put a framework in place for specifying those rules -- all without making code-level or class-level changes.

Before getting to those, let me give you some strong advice: You should *always* use `DynaValidatorForm` in your form beans. Because it acts just like the Struts `DynaActionForm` when no validation rules are set up, it's a drop-in replacement. More importantly, though, you can always add validation rules later and not have to make changes to your form beans. For this reason (unless I am using a custom `ActionForm` implementation), I try to use `DynaActionForm` in all my form beans.

Adding validation rules

Now we come to the fun part. Open up `WEB-INF/validation.xml`. This is the second of the two Validator-specific configuration files I mentioned earlier (the first was

validator-rules.xml, which specified general validation rules). The validation.xml file contains mappings between specific fields in your forms (defined in struts-config.xml) and general validation rules (in validator-rules.xml). Open up validation.xml and find the `formset` element. Within that element, add the code in Listing 14 (it won't make sense yet, but I'll explain it all to you shortly):

Listing 14. Code for the formset element

```
<form name="ValidationForm">
  <field property="username"
    depends="required,minlength">
    <arg0 key="prompt.username" />
    <arg1 key="{var:minlength}" name="minlength"
      resource="false" />
    <var>
      <var-name>minlength</var-name>
      <var-value>6</var-value>
    </var>
  </field>
  <field property="password"
    depends="required,minlength">
    <arg0 key="prompt.password" />
    <arg1 key="{var:minlength}" name="minlength"
      resource="false" />
    <var>
      <var-name>minlength</var-name>
      <var-value>8</var-value>
    </var>
  </field>
</form>
```

First, the `form` element indicates that a new form is being detailed; the `name` attribute identifies that form. The name here should match up with the `form-bean` element in `struts-config.xml`, as you would expect. With the form listed, you now need to specify rules for each field you want to validate.

In this case, I've used two of the simplest, and most common, rules on the `username` and `password` fields. The first of these is `required`, and it -- as well as any other rules -- is indicated through the `depends` attribute of the `field` element. You just list the rules you want to use, one after another, separated by commas. Then, for each rule, you supply some data to the validator for use. For example, the `required` rule needs a single argument; this argument is the name of the field and is displayed in the error message generated if the field has no value. By using the `arg0` element, you can supply that name (in this case, `prompt.username` and `prompt.password`, pulled straight from the trusty `MessageResources.properties` file). Starting to see how all of this fits together?

In addition to the `required` rule, I'm also using the `minlength` rule. (Note that these rules *are* case-sensitive! If you mistype one, it will simply be ignored, which can be quite frustrating.) This rule needs more than just an argument; it needs a variable indicating what the minimum acceptable length for a field is. This is passed into the `minlength` rule through the `var`, `var-name`, and `var-value` elements.

This should again be pretty clear from the code, so I'll leave it to you to figure out exactly what's going on here.

Finally, another argument is required -- this time for any error message that the `minlength` rule may need to generate. Because `arg0` is used by the `required` rule, just bump that up by one to `arg1` and supply the `minlength` rule the minimum number of characters required. Note how the variable is referenced here. The `resource="false"` statement tells the validator to pull this data from the current file, rather than trying to grab data from the application's resource bundle (`MessageResources.properties`).

At this point, your application is going to require values for the username and password fields, as well as minimum lengths for any data entered into those fields (note that each field has a different minimum length). Pretty nifty! And even though the rule definitions in `validation.xml` are pretty verbose, it's clear what each does.

Using built-in Validator rules

The Struts Validator comes with quite a few useful built-in rules; you can check data types (`integer`, `data`, `byte`, etc.) and ranges of data (`range`). You've already seen the `minlength` rule, and there's also a `maxlength` rule. You can even use the `creditCard` rule to ensure that digits match the credit card numeric format. I'm going to use two other rules -- `email` and `url` -- to ensure correct data on the validation test form.

Here are some new rule definitions for `validation.xml`:

```
<field property="email"
  depends="email">
  <arg0 key="prompt.email" />
</field>
<field property="url"
  depends="url">
  <arg0 key="prompt.url" />
</field>
```

The first of these require the data in the E-Mail field to be a valid address (isn't it nice to let Struts deal with this?). The second ensures that the URL is valid.

(Note: At least in my own testing, Struts did not produce a JavaScript version of the URL-validation method. This means that if all of the form's fields other than the URL field contain valid data, it will pass client-side validation but fail server-side validation. This isn't the worst thing that could happen, as you still get the URL field validated, but I expect it to be corrected in future versions of Struts. In any case, there's certainly no reason to stop using the `url` rule.)

Using the mask rule

There will be many cases in which you have textual data that needs to be in a particular format; however, Struts can hardly account for every possible one of these formats: phone numbers, e-mail addresses, URLs, zip codes, driver's license numbers ... the list just goes on and on. So, Struts takes care of the most common (`email` and `url`) and provides the `mask` rule for everything else. You can use the `mask` rule to specify your own text pattern, using regular expressions.

If you're comfortable with regular expressions, this should be a piece of cake. (An introduction to regular expressions is well beyond the scope of this tutorial. See the [Resources](#) for more on regular expression literature if you need help.) Here's how you might define the rule for the `phone` field:

```
<field property="phone"
  depends="required,mask">
  <arg0 key="prompt.phone" />
  <var>
    <var-name>mask</var-name>
    <var-value>^\((?\d{3})\)?[-| ]?(?(\d{3})[-| ]?(?(\d{4}))$</var-value>
  </var>
</field>
```

This is a pretty common pattern to use, and it allows most of the popular variations on phone formats to be entered. If an invalid form is used, the Validator will spit out a message about invalid data.

However, you could easily use this same mask in several places. Rather than having to define it each time it's used, move the mask definition up into a *global constant*. Make the following addition to your `validation.xml` file within the `global` element (near the top of the file):

```
<global>
  <constant>
    <constant-name>phone</constant-name>
    <constant-value>^\((?\d{3})\)?[-| ]?(?(\d{3})[-| ]?(?(\d{4}))$</constant-value>
  </constant>
</global>
```

Now change the entry for the `phone` field to look like this, using this new constant mask value:

```
<field property="phone"
  depends="required,mask">
  <arg0 key="prompt.phone" />
  <var>
    <var-name>mask</var-name>
    <var-value>${phone}</var-value>
```

```
</var>  
</field>
```

Now you can reuse this same pattern in other forms throughout your application's validation rules.

Finally, note that I haven't given you a comprehensive list of Validator's built-in rules. That's intentional; these are changing constantly, and support for new rules is coming along all the time. For the latest set of rules, check out the Struts Validator development guide online (see [Resources](#) for a link). Who knows what interesting rules will be available by the time you read this?

Supporting validation in JSPs

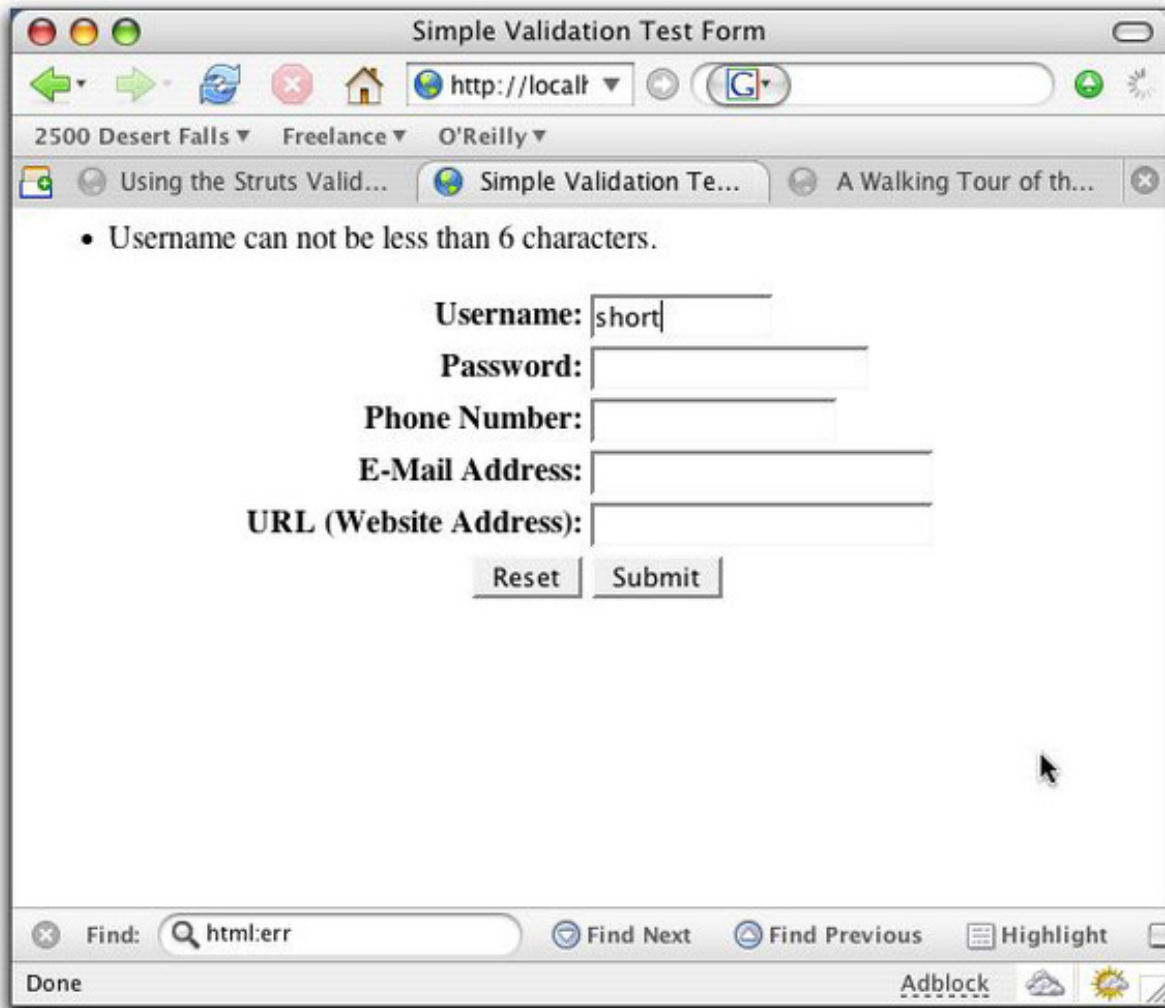
These rules are all well and good, but there's still a significant problem. Your JSP pages have no way of reporting the errors! For example, if you redeployed your application at this point, you would find that submitting the validation form with invalid values would *not* take you to success.jsp. That's good and a step in the right direction. However, it's also incredibly frustrating; the form just keeps reappearing in the browser, with no indication of what went wrong.

Open pages/test-validation.jsp, and let's take care of that problem. First, you need to provide a place for errors to show up when they occur. This is nicely handled by the `html:errors` element. Insert the following right before your `html:form` element:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>  
<%@ taglib uri="/tags/struts-html" prefix="html" %>  
  
<html:xhtml />  
  
<html>  
  <head>  
    <title><bean:message key="valid.title" /></title>  
  </head>  
  
  <html:errors />  
  
  <html:form action="/SubmitValid" focus="username">
```

If you redeploy at this point and try and enter invalid data, you will start to see error messages, as shown in Figure 4:

Figure 4. Submitting the form with invalid data now triggers errors messages



Again, this is moving closer toward really solid validation. However, it's still server-side; you get that annoying flash while the form submits and then have to wait for the server to respond. The goal here was validation encoded in JavaScript (or, even better, JavaScript that you didn't have to write manually for each form). Happily, the Validator lets you convert its server-side code to client-side JavaScript with just a few additional tags:

```
</html:form>  
<html:javascript formName="ValidationForm" cdata="false" />  
</body>
```

Not much there, huh? It's a single line of code, but it tells Struts to insert all the JavaScript to support client-side validation of your form. Also note the very important `cdata` attribute. You *must* set this to `false`, or client-side validation isn't going to work for you. If it's set to `true` (the default), the resulting HTML that Struts serves

will have the JavaScript enclosed in `<![CDATA[and]]>` tags. For reasons that I admit I'm unclear on, most modern browsers (Mozilla Firefox and Safari most notable among them) ignore JavaScript that is so enclosed, and client-side validation won't execute.

At this stage, I should point out that even if you forget the `cdata=false` portion, you're still going to get validation, albeit on the server-side. The Validator will validate the data on the server-side in either case, so you've got a backup if you forget this attribute, or even if your users turn off JavaScript or get sneaky and bypass it. It's just another nice side-effect of using the Validator -- it really does try and catch every possible error.

Testing out the completed application

With JavaScript enabled, redeploy your application one last time and enter all sorts of invalid data. Click **Submit**, and you'll get a great client-side list of errors (as shown in Figures 5, 6, and 7). This may be one of the only times you'll be excited to see error messages!

Figure 5. Here, the password is too short



Figure 6. In this case, the phone number is improperly formatted



Figure 7. An invalid e-mail is indeed seen as invalid



These messages are handled sequentially, and it's largely up to the Validator engine to decide how many to tackle at once. In other words, you might get a single error stating that several fields left blank must be filled in. Once you click **OK** and correct that error, though, a different error (such as invalid formatting on a phone number) will crop up. The Validator will always only present you with one dialog box, but that box may contain multiple errors. However the Validator chooses to present these errors, though, it will ensure that all validation rules are followed before letting the user continue.

Section 7. Summary

Wrapping up the Validator

This may have seemed like a whole lot of configuration and not much programming.

I'm as much a coder as the next person, but sometimes you have to spend a lot of time configuring so you can spend a little time coding. Without the Validator -- or a component like it -- you'll have to write Java or JavaScript code to manually validate each and every input box on your Struts forms. Even someone who loves to code hates to write validation handlers, and especially hates writing them over and over again.

You should now have a working Validator setup, and a nice test environment built from the Struts example application and the validation rules detailed in this tutorial. I often use this exact setup when I install Tomcat, Struts, and the Validator on a new machine; it saves me trouble in trying to track down errors. Additionally, you can use this same configuration for your existing -- and future -- Web applications.

Along the way, you should have picked up some real familiarity with Struts and the Validator component, and in particular with their configuration files. Even when you're not working with the Validator component, this should help you master your Struts installation.

However you use this tutorial, I hope it takes your Struts development (and validation) to the next level. Enjoy!

Feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like *developerWorks* to cover.

For technical questions or comments about the content of this tutorial, contact the author, Brett McLaughlin, at brett@newInstance.com.

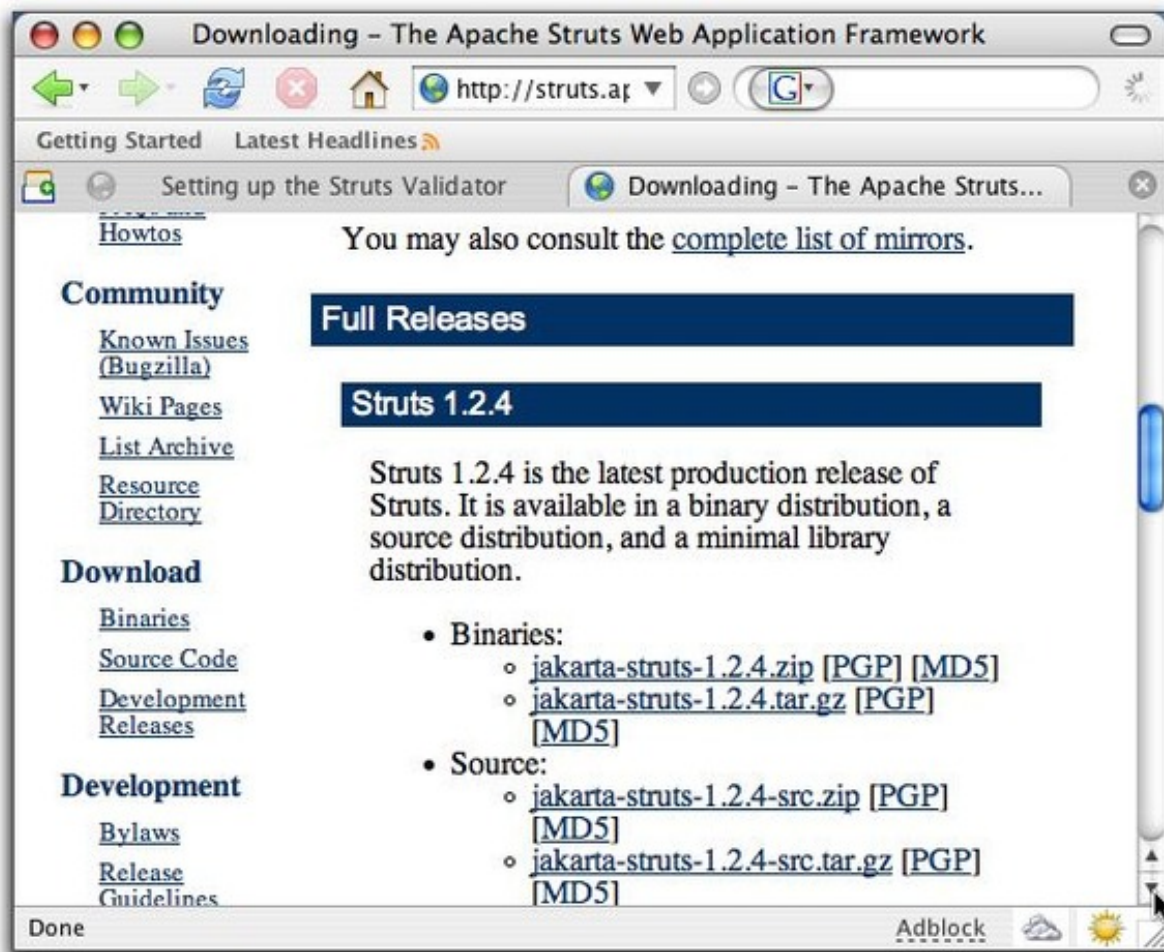
Section 8. Appendix: Installing Struts on Tomcat 5.5.x

Downloading Struts

While this isn't really a tutorial about Struts itself, I want to briefly run through a simple Struts installation. While I hope that you already have some basic Struts experience, I realize that many of you may be trying out Struts by way of this tutorial. If you're in that category, walk through these steps to get Struts and the example application up and running.

1. Start out by going to the [Struts Web site](#). On the left side of the page, you'll see a **Downloads** heading (you may have to scroll down a bit); underneath that, select **Binaries**.
2. Choose an alternate mirror if you like (I generally use the default mirror site), and find the first entry under **Full Releases**. As I write this, that's Struts 1.2.4 (see Figure 8).
3. Select the download appropriate for your system, and download the file to your local machine.

Figure 8. Download the latest full binary release of Struts

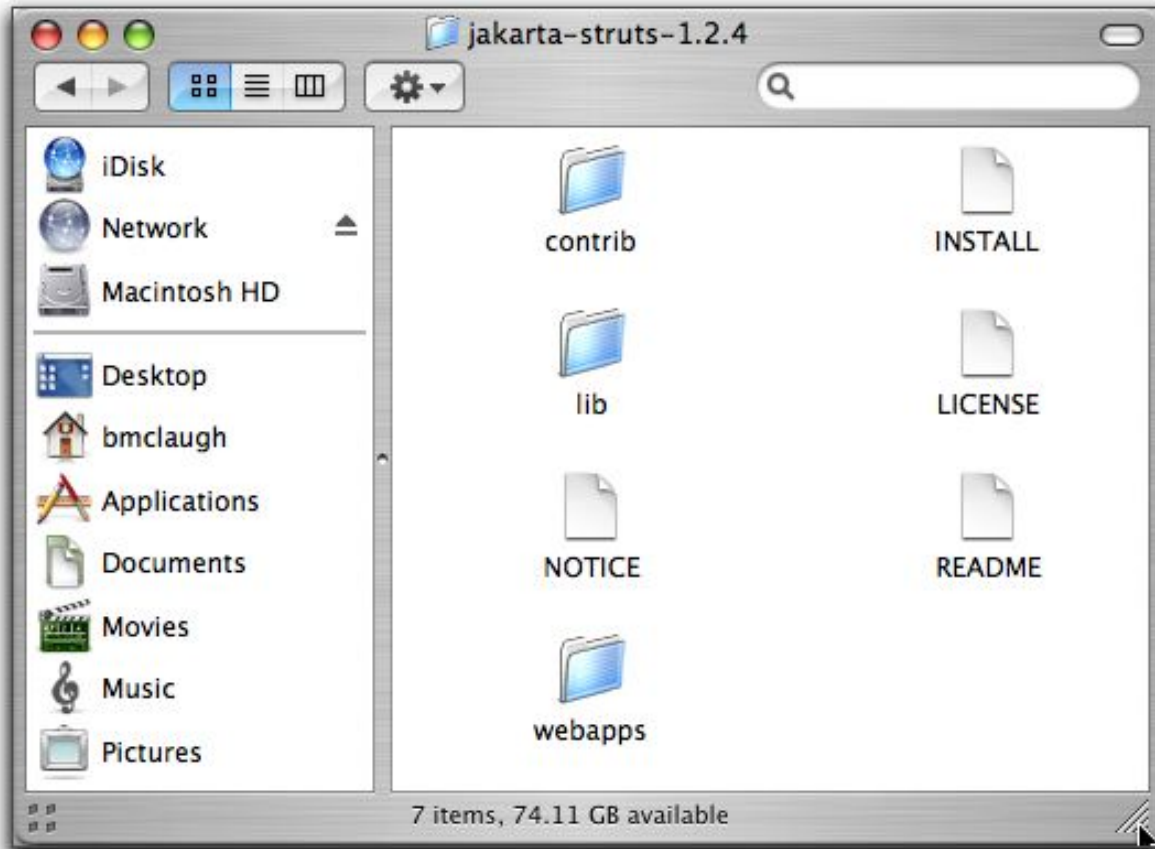


4. Once you've downloaded the file, go ahead and expand the archive; you should get a directory named something like jakarta-struts-1.2.4. At this point, you can delete the archive (and, if you created a TAR file in the process of decompression, delete that as well).

Installing Struts

1. Move your newly created Struts folder into a directory where you store your Java projects. I moved it into /usr/local/java on my system, resulting in /usr/local/java/jakarta-struts-1.2.4. Go ahead and navigate into that folder; it should look something like Figure 9:

Figure 9. There's really not much to a Struts binary release



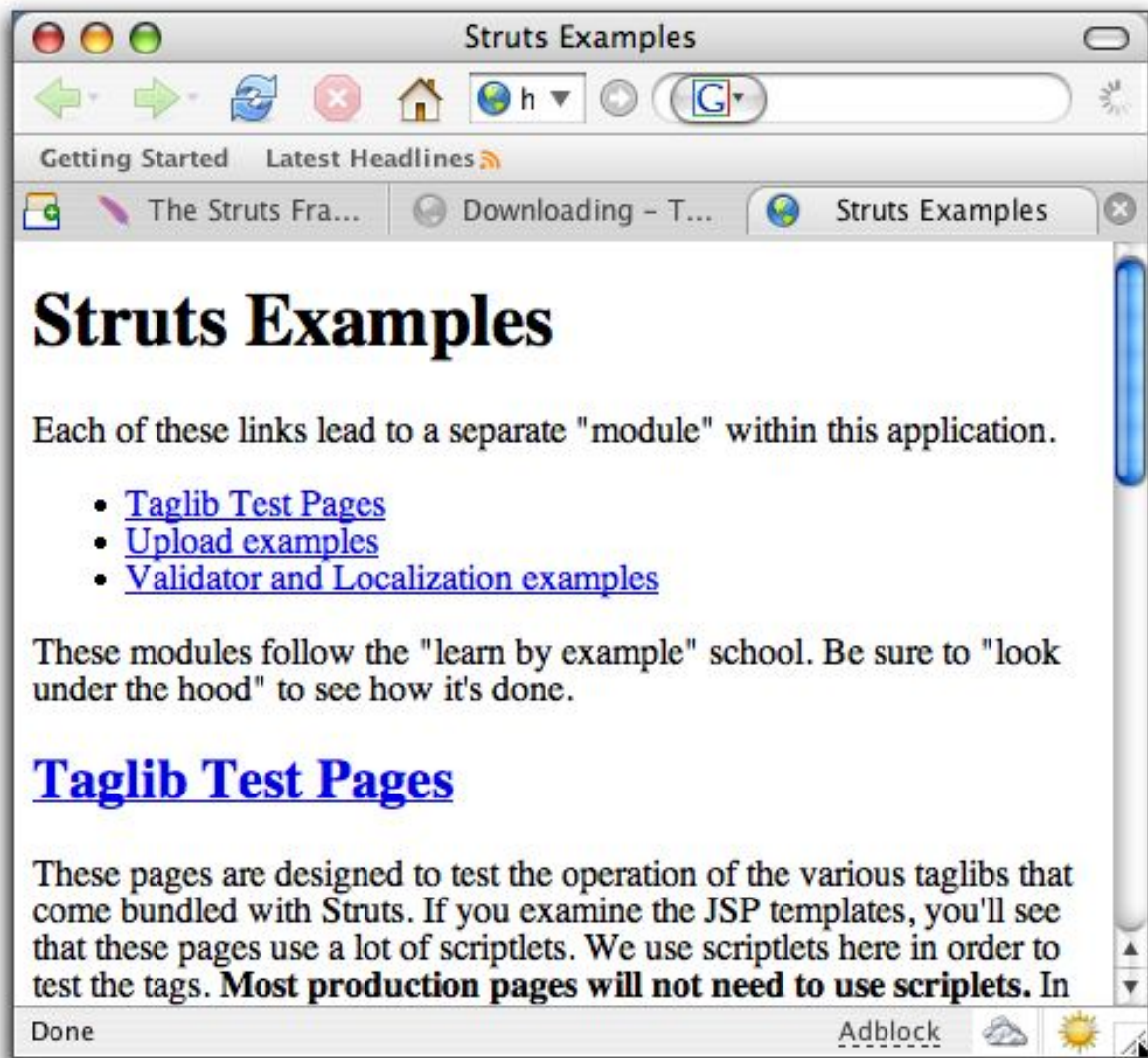
2. Drill down into the webapps folder; you should see several WAR files (there are five of these in Struts 1.2.4). Copy all of these files to your Tomcat webapps directory. If Tomcat isn't running, start it up. Tomcat will expand each of these WAR files, and automatically load the applications.
3. To ensure that everything is working properly, navigate to <http://localhost:8080/struts-example>.

Note: I'm assuming a default Tomcat installation. If you've got Tomcat running on a different host, you'll obviously need to replace `localhost` with your server's

hostname. If you don't have WARs automatically deployed, or if you use a different directory for your contexts, then you're no doubt comfortable enough with Tomcat to make the appropriate changes.

You should see something that looks like Figure 10:

Figure 10. The struts-example Web application is an easy way to ensure Struts is working



4. You should click on a few links and make sure everything looks right; no error pages or odd messages should appear. If this all looks correct, then you've got a working Struts installation!

Downloads

Description	Name	Size	Download method
Sample code	j-strutsvalcode.zip	2 MB	HTTP

[Information about download methods](#)

Resources

Learn

- For all you ever wanted to know about regular expressions, try out [Mastering Regular Expressions](#), Jeffrey Friedl (O'Reilly, 2002).
- You can read up on how to [install Struts on various servlet engines](#) at the Struts Web site.
- The Struts Validator [Developer's Guide](#) is a great resource for learning more about Struts and the Validator. Here you can find the most recent list of built-in Validator rules.
- There are two great O'Reilly books on Struts: [Programming Jakarta Struts](#), Chuck Cavaness (2004) and [Jakarta Struts Cookbook](#), Bill Siggelkow (2005). Both cover the Validator, as well as declarative programming, in detail.
- I keep a copy of the [Regular Expressions Pocket Guide](#), Tony Stubblebine (O'Reilly, 2003) nearby for help writing masks.
- developerWorks has published a number of Struts-related articles and tutorials:
 - "[Best practices for Struts development](#)," Palaniyappan Thiagarajan and Pagadala Suresh (June 2004)
 - "[Developing Struts with Easy Struts for Eclipse](#)," Nancy Chen Junhua (April 2004)
 - "[Integrating Struts, Tiles, and JavaServer Faces](#)," Srikanth Shenoy and Nithin Mallya (September 2003)
 - "[Struts, an open source MVC implementation](#)," Malcolm Davis (February 2001)
- Find out how [apply validation to form fields in WebSphere](#).
- You'll find articles about every aspect of Java programming, including all the concepts covered in this tutorial, in the developerWorks [Java technology zone](#).

Get products and technologies

- You can download Apache projects, and find out more about the Apache Software Foundation, at the [Apache Web site](#).
- The Apache Web server lives online at its own [Web site](#). Although I didn't talk about it in this tutorial, I always serve HTML pages with Apache, and pass through Java requests to Tomcat.
- Struts resides [online as a top-level Apache project](#), and is a well-documented open source offering.

- IBM offers [Intelligent Forms Processing](#) for OCR technologies.

About the author

Brett McLaughlin



Brett McLaughlin has worked in computers since the Logo days. (Remember the little triangle?) In recent years, he's become one of the best-known authors and programmers in the Java technology and XML communities. He's worked for Nextel Communications, implementing complex enterprise systems; at Lutris Technologies, actually writing application servers; and, most recently, at O'Reilly Media, Inc., where he continues to write and edit books that matter. His most recent book, [Java 1.5 Tiger: A Developer's Notebook](#), was the first book available on the newest version of Java technology, and his classic [Java and XML](#) remains one of the definitive works on using XML technologies in the Java language.