

Struts Quick Start Tutorial

Getting started with Struts, skip the fluff, just the facts!

This chapter is a tutorial that covers getting started with Struts—just the basics, nothing more, nothing less. This tutorial assumes knowledge of Java, JDBC, Servlets, J2EE (with regards to Web applications) and JSP. Although you can follow along if you are not an expert in all of the above, some knowledge of each is assumed.

Instead of breaking the chapters into a myriad of single topics, which are in depth ad nauseum, we treat Struts in a holistic manner, minus the beads and crystals. The focus of this chapter is to skim briefly over many topics to give you a feel for the full breadth of Struts. Later chapters will delve deeply into many topics in detail.

In this chapter, you will cover:

- The struts configuration
- Writing Actions
- Working with Struts Custom tags
- Setting up datasource
- Handling exceptions
- Displaying objects in a JSP

In this chapter you will perform the following steps:

1. Download Struts
2. Setup a J2EE web application project that uses Struts
3. Write your first Action
4. Write your first “forward”
5. Configure the Action and forward in the Struts configuration file
6. Run and Test your first Struts application.
7. Debugging Struts-Config.xml with the *Struts Console*
8. Add Logging support with Log4J and commons logging.
9. Write your first ActionForm
10. Write your first input view (JSP page)
11. Update the Action to handle the form, and cancel button
12. Setup the database pooling with Struts
13. Declaratively Handle Exception in the Struts Config file
14. Display an Object with Struts Custom Tags

Download Struts

The first step in getting started with Struts is to download the Struts framework. The Struts home page is located at <http://jakarta.apache.org/struts/index.html>. You can find online documentation at the Struts home page. However, to download Struts you need to go to the Jakarta Download page at <http://jakarta.apache.org/site/binindex.cgi>. Since all of the Jakarta download links are on the same page, search for “Struts” on this page. Look for the link that looks like this:

Struts [KEYS](#)

- [1.1 zip PGP MD5](#)
- [1.1 tar.gz PGP MD5](#)

Download either compressed file.

One of the best forms of documentation on Struts is the source. Download the source from <http://jakarta.apache.org/site/sourceindex.cgi>. Once you have both the source and binaries downloaded, extract them. (WinZip works well for Windows users.) This tutorial will assume that you have extracted the files to c:\tools\jakarta-struts-1.1-src and c:\tools\jakarta-struts-1.1. If you are using another drive, directory, or *n?x (UNIX or Linux), adjust accordingly.

Set up a J2EE Web Application Project That Uses Struts

Struts ships with a started web application archive file (WAR file) called *struts-blank.war*. The *struts-blank.war* file has all of the configuration files, tag library descriptor files (tld files) and JAR files that you need to start using Struts. The *struts-blank.war* file includes support for Tiles and the Validator framework. You can find the *struts-blank.war* file under `C:\tools\jakarta-struts-1.1\webapps`.

1. A war file is the same format as a ZIP file. Extract the *struts-blank.war* file to a directory called `c:\strutsTutorial` (adjust if `*n?x`). When you are done, you should have a directory structure as follows:

```
C: .
|---META-INF
|---pages
|---WEB-INF
|   |---classes
|   |   |--resources
|   |---lib
|   |---src
|       |---java
|           |---resources
```

The blank war file ships with an Ant build script under `WEB-INF/src`. The structure of the extracted directory mimics the structure of a deployed web application.

2. In order for the *build.xml* file to work, you need to modify it to point to the jar file that contains the Servlet API and the jar file that points to the JDBC API from your application server.

For example, if you had Tomcat 5 installed under `c:\tomcat5`, then you would need to modify the *servlet.jar* property as follows:

```
<property name="servlet.jar"
          value="C:/tomcat5/common/lib/servlet-api.jar"/>
```

Tip: Tomcat is a Servlet container that supports JSP. If you are new to web development in Java, there are several very good books on Java web development. You will need to know about JSP, Servlets and web applications to get the most out of this chapter and this book. If you are new to Java web development, try this tutorial: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/WebApp.html#wp76431>.

3. If you are using an IDE (Eclipse, NetBeans, JBuilder, WSAD, etc.), set up a new IDE project pointing to `C:\strutsTutorial\WEB-INF\src\java` as the source directory, add your application server's *servlet.jar* file (*servlet-api.jar* for tomcat) and all the jar files from `C:\strutsTutorial\WEB-INF\lib`.

Write Your First Action

Actions respond to requests. When you write an Action, you subclass `org.apache.struts.action.Action` and override the `execute` method.

The `execute` method returns an `ActionForward`. You can think of an `ActionForward` as an output view.

The `execute` method takes four arguments: an `ActionMapping`, `ActionForm`, `HttpServletRequest` and an `HttpServletResponse` (respectively).

The `ActionMapping` is the object manifestation of the XML element used to configure an Action in the Struts configuration file. The `ActionMapping` contains a group of `ActionForwards` associated with the current action.

For now, ignore the `ActionForm`; we will cover it later. (It is assumed that you are familiar with `HttpServletRequest` and `HttpServletResponse` already.)

Go to `strutsTutorial\WEB-INF\src\java` and add the package directory `strutsTutorial`. In the `strutsTutorial` directory, add the class `UserRegistration` as follows:

```
package strutsTutorial;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class UserRegistrationAction extends Action {

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception {

        return mapping.findForward("success");
    }

}
```

Notice that this Action forwards to the output view called `success`. That output view, `ActionForward`, will be a plain old JSP. Let's add that JSP.

Write Your First “Forward”

Your first forward will be a JSP page that notifies the user that their registration was successful. Add a JSP page to `c:\strutsTutorial` called `regSuccess.jsp` with the following content:

```
<html>
<head>
<title>
User Registration Was Successful!
</title>
</head>

<body>
<h1>User Registration Was Successful!</h1>
</body>
</html>
```

The forward is the output view. The Action will forward to this JSP by looking up a forward called `success`. Thus, we need to associate this output view JSP to a forward called `success`.

Configure the Action and Forward in the Struts Configuration File

Now that we have written our first Action and our first Forward, we need to wire them together. To wire them together we need to modify the Struts configuration file. The Struts configuration file location is specified by the config init parameter for the Struts ActionServlet located in the web.xml file. This was done for us already by the authors of the blank.war starter application. Here is what that entry looks like:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    org.apache.struts.action.ActionServlet
  </servlet-class>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
  </init-param>
  ...
  <load-on-startup>2</load-on-startup>
</servlet>
```

Thus, you can see that the blank.war's web.xml uses WEB-INF/struts-config.xml file as the Struts configuration file.

Follow these steps to add the success forward:

1. Open the c:\strutsTutorial\WEB-INF\struts-config.xml file.
2. Look for an element called action-mappings.
3. Add an action element under action-mappings as shown below.

```
<action path="/userRegistration"
        type="strutsTutorial.UserRegistrationAction">
    <forward name="success" path="/regSuccess.jsp"/>
</action>
```

The above associates the incoming path /userRegistration with the Action handler you wrote earlier, strutsTutorial.UserRegistrationAction.

Whenever this web application gets a request with /userRegistration.do, the execute method of the strutsTutorial.UserRegistrationAction class will be invoked. The web.xml file maps request that end in *.do to the Struts ActionServlet. Since the web.xml file was provided for us, you will not need to edit it. Here is the mapping for in the web.xml file for reference:

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

The Struts ActionServlet will invoke our action based on the path attribute of the above action mapping. The ActionServlet will handle all requests that end in *.do. You may recall that our Action looks up and returns a forward called success. The forward element maps the success forward to the regSuccess.jsp file that you just created in the last section.

The ActionMapping that gets passed to the execute method of the Action handler is the object representation of the action mapping you just added in the Struts config file.

Run and Test Your First Struts Application

The blank.war file ships with a started Ant script that will build and deploy the web application.

If you are new to Ant, then today is a good day to get up to speed with it. Ant is a build system from Jakarta. Struts uses a lot of Jakarta projects. Most Jakarta projects use Ant. Ant is also a Jakarta project. (Technically, it used to be a Jakarta project, and it was promoted to a top level project at the 1.5 release.) You can learn more about Ant and read documentation at <http://ant.apache.org/>. You can download Ant at <http://www.apache.org/dist/ant/>. Also, you can find an install guide for Ant at <http://ant.apache.org/manual/installlist.html>.

Technically, you do not have to use Ant to continue on with this tutorial, but it will make things easier for you. It's up to you. If you are not using Ant, now is a good time to start. Read <http://ant.apache.org/manual/usinglist.html> to start using Ant after you install it.

If you are using the Ant build.xml file that ships with the blank.war (look under WEB-INF/src), you will need to add the `{servlet.jar}` file to the compile.classpath as follows:

```
<path id="compile.classpath">
  <pathelement path="lib/commons-beanutils.jar"/>
  <pathelement path="lib/commons-digester.jar"/>
  <pathelement path="lib/struts.jar"/>
  <pathelement path="classes"/>
  <pathelement path="{classpath}"/>
  <pathelement path="{servlet.jar}"/>
</path>
```

Notice the addition of the `<pathelement path="{servlet.jar}"/>`. The compile classpath is used by the compile target.

After you add the pathelement, change the project.distname property to strutsTutorial as follows:

```
<property name="project.distname" value="strutsTutorial"/>
```

Go ahead and run the Ant build script as follows:

```
C:\strutsTutorial\WEB-INF\src> ant
```

If things go well, you should see the message BUILD SUCCESSFUL. Once you run the Ant build script with the default target, you should get a war file in your `c:\projects\lib` directory called `strutsTutorial.war`. Deploy this war file to your application server under the web context `strutsTutorial`. You will need to refer to your application server manual for more details on how to do this. For Tomcat and Resin, this is a simple matter of copying the war file to Tomcat's or Resin's `home-dir/webapps` directory. The `webapps` directory is under the server directory.

If you are not Ant savvy, you can simulate what this ant script does by setting your IDE to output the binary files to C:\strutsTutorial\WEB-INF\classes, and then zipping up the c:\strutsTutorial directory into a file called strutsTutorial.war.

Now that you have built and deployed your web application, test your new Action/forward combination by going to <http://localhost:8080/strutsTutorial/userRegistration.do>. You should see the following:

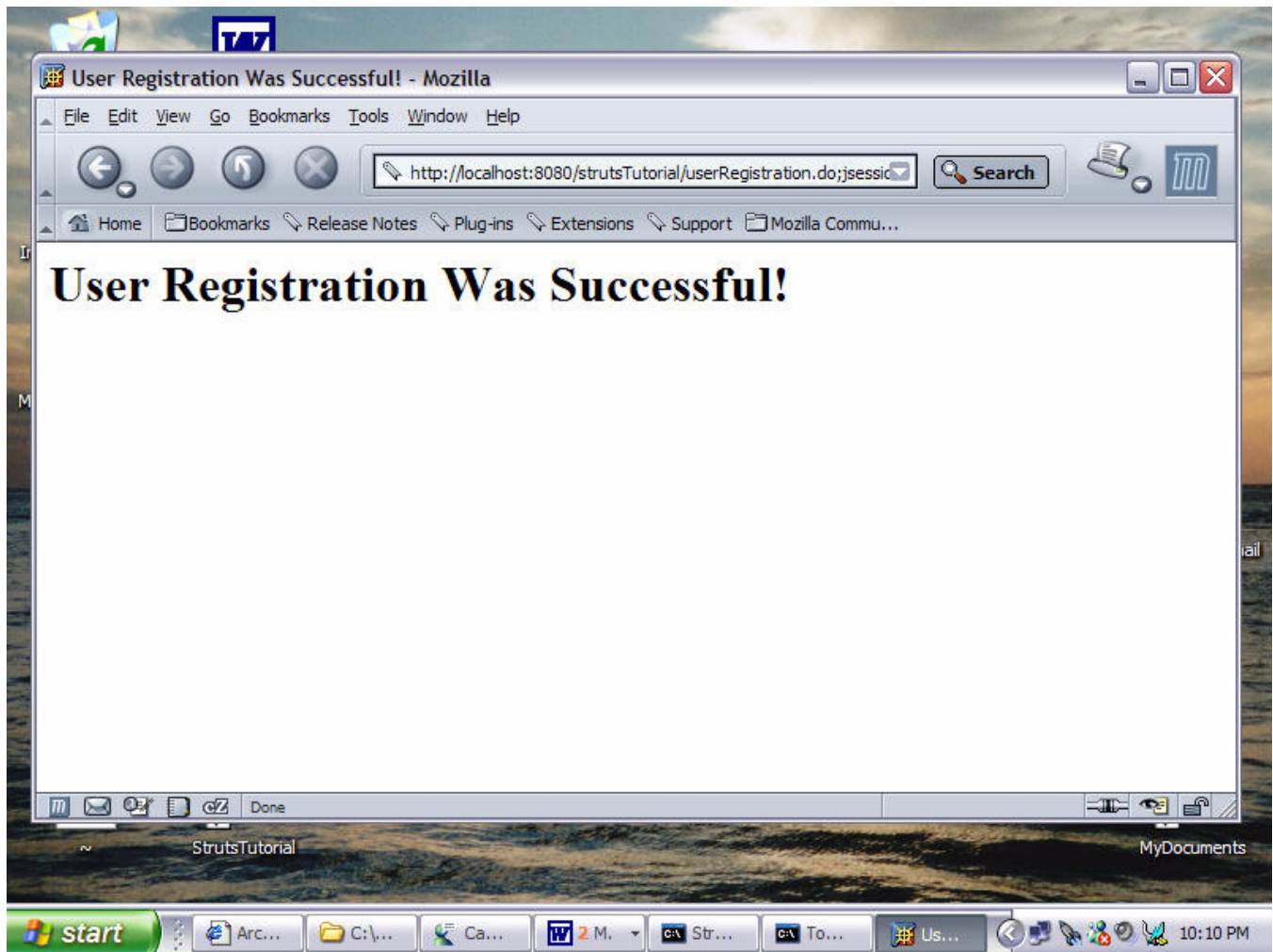


Figure 1.1 Running The Action for the first time

Congratulations! You have just written your first Struts Action. Now, admittedly this Action does not do much.

At this point, you may be having troubles. The most obvious problem is probably a misconfigured struts-config.xml file. There are two ways to solve this.

Debug Struts-Config.xml with the *Struts Console*

If you are new to XML, it may be a little hard to edit the `struts-config.xml` file. If you are having troubles with the `struts-config.xml` file, you should download the *Struts Console*. The *Struts Console* is a Swing based `struts-config.xml` editor; it provides a GUI for editing `struts-config` files. The *Struts Console* works as a plugin for JBuilder, NetBeans, Eclipse, IntelliJ and more. The *Struts Console* can be found at <http://www.jamesholmes.com/struts/console/>; follow the install instructions at the site. If you have a problem, you can edit the `struts-config` file with *Struts Console*. If there is a problem with the `struts-config.xml` file, the *Struts Console* will take you to the line / column of text that is having the problem. For example, here is an example of editing a `struts-config` file with a malformed XML attribute:

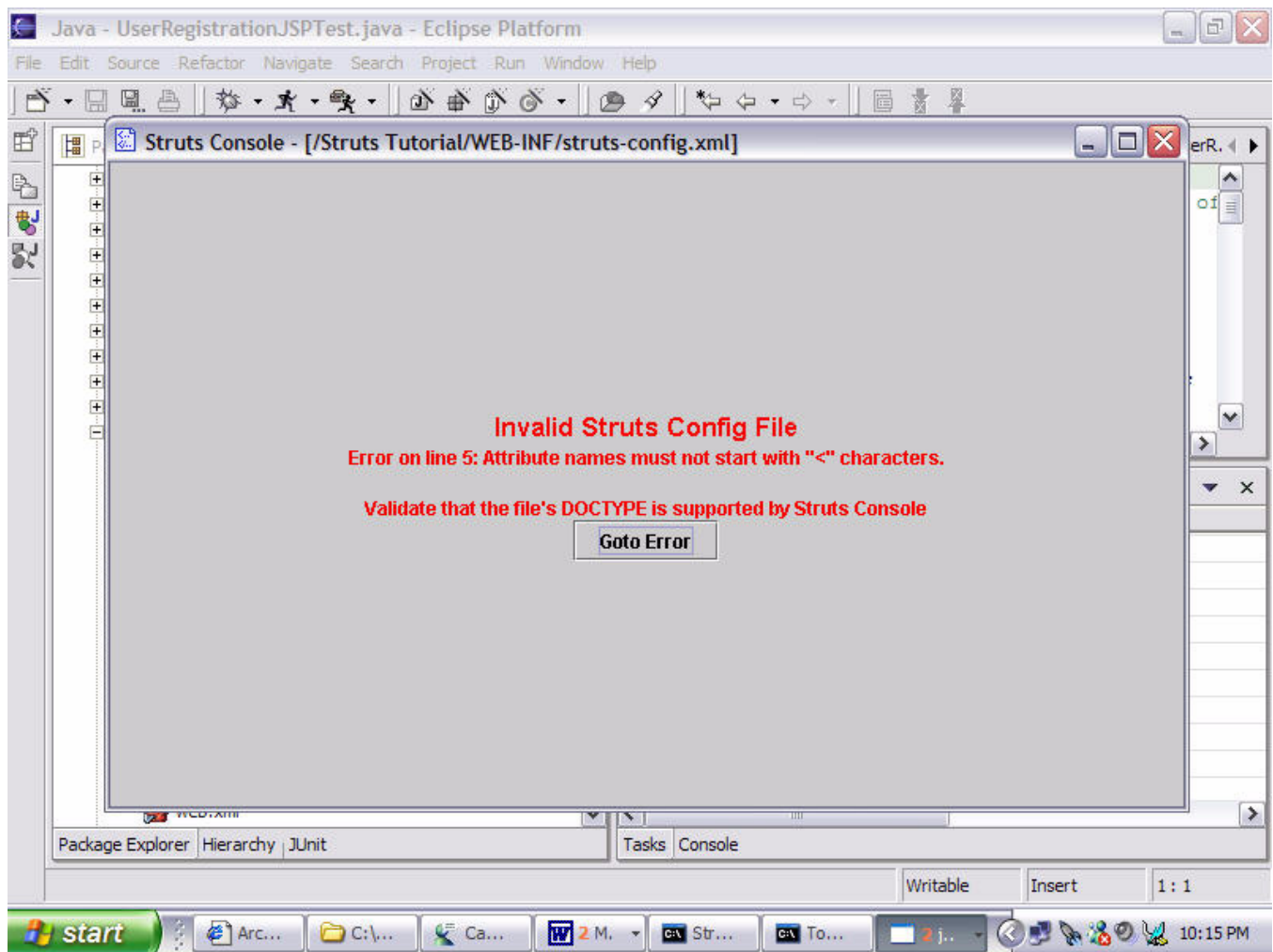


Figure 1.2 Running *Struts Console* against a malformed `struts-config.xml` file

Notice the Goto Error button. Clicking the button takes you to the exact line in the `struts-config.xml` file that is having the problem.

Once everything is fixed, you can view and edit the `struts-config.xml` file with the *Struts Console* as follows:

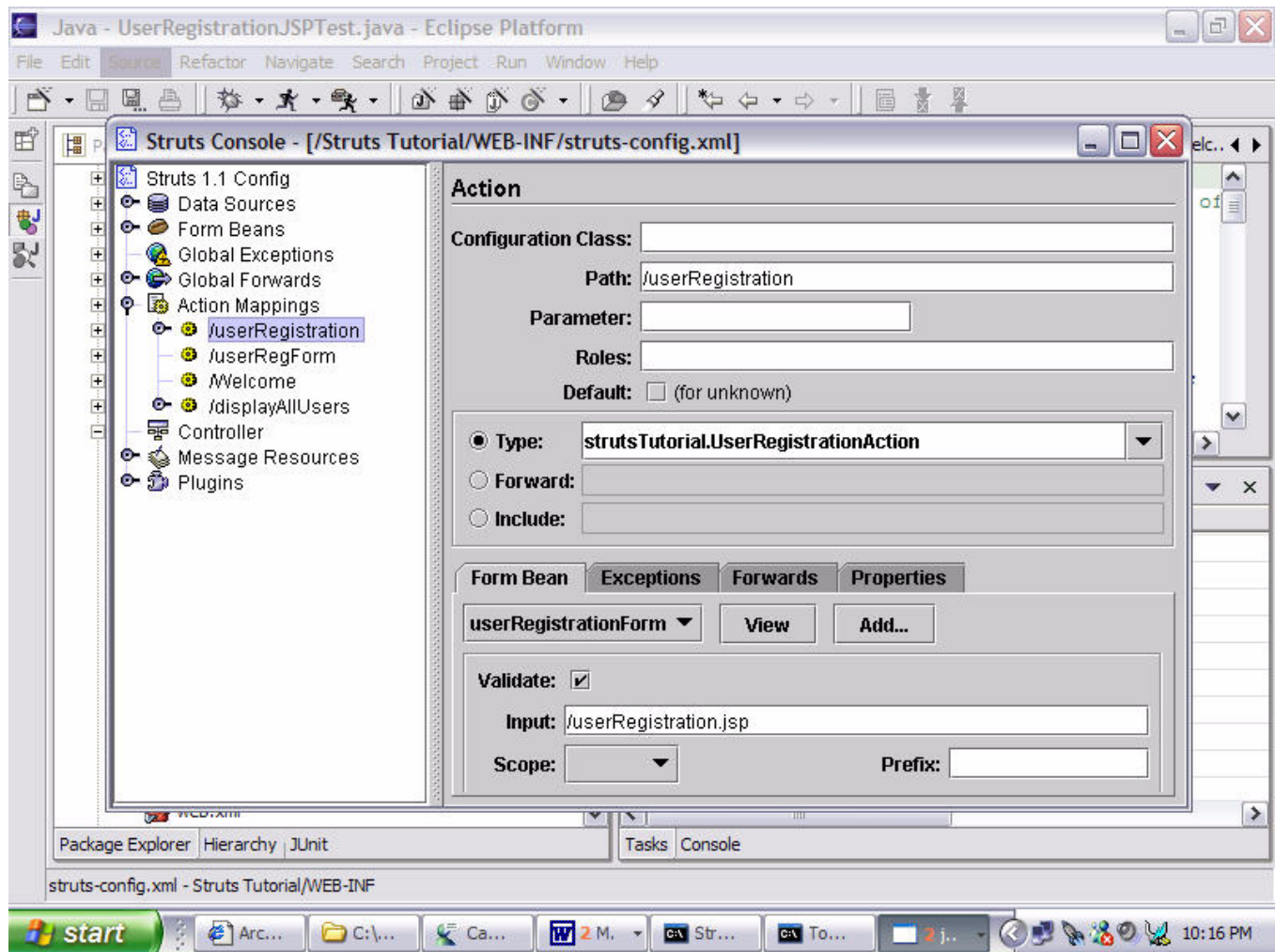


Figure 1.3 Running *Struts Console* in a happy world

The figure above shows editing `struts-config.xml` and inspecting the `userRegistration` action that you just configured.

Personally, I only use the *Struts Console* if there is a problem or I want to validate my `struts-config.xml` file without launching the application server. I prefer editing the `struts-config.xml` with a text editor, but find that the *Struts Console* comes in handy when there is a problem. In addition to mentoring new Struts developers and doing development myself, I teach a Struts course. I have found that *Struts Console* is extremely valuable to new Struts developers. Students (and new Struts developers) can easily make a small mistake that will cause the config file to fail. I can stare for a long time at a `struts-config.xml` file, and not find a “one off” error. Most of these errors, you will not make once you are a seasoned Struts developer, but they can be very hard to diagnose without the *Struts Console* when you are first getting started.

Another debugging technique is to use common logging to debug the application at runtime.

Add Logging Support with Log4J and Commons Logging

You may wonder why we dedicate a whole section to logging. Well to put it simply, when you are new to Struts, you will need to do more debugging, and logging can facilitate your debugging sessions dramatically.

The Struts framework uses Commons Logging throughout. Logging is a good way to learn what Struts does at runtime, and it helps you to debug problems. The Commons Logging framework works with many logging systems; mainly Java Logging that ships with JDK 1.4 and Log4J.

Using Struts without logging can be like driving in the fog with your bright lights, especially when something goes wrong. You will get a much better understanding how Struts works by examining the logs. Logging can be expensive. Log4J allows you to easily turn off logging at runtime.

Log4J is a full-featured logging system. It is easy to set up and use with Struts. You need to do several things:

1. Download Log4J
2. Unzip the Log4J distribution
3. Copy the log4j.jar file to c:\strutsTutorial\WEB-INF\lib
4. Create a log4j.properties file
5. Start using logging in our own classes

Like Struts, Log4J is an Apache Jakarta project. The Log4J home page is at <http://jakarta.apache.org/log4j/docs/index.html>. You can download Log4J from <http://jakarta.apache.org/site/binindex.cgi>. Search for Log4J on this page. Look for the link that looks like:

Log4j [KEYS](#)

- [1.2.8 zip PGP MD5](#)
- [1.2.8 tar.gz PGP MD5](#)

Click on the 1.2.8 ZIP file link. Download this file to c:\tools, and adjust accordingly for different drives, directories and operating systems (like *n?x). Copy the log4j-1.2.8.jar file located in C:\tools\jakarta-log4j-1.2.8\dist\lib to c:\strutsTutorial\WEB-INF\lib.

Now that you have the jar file in the right location you need to add log4j.properties files so that the web application classloader can find it. The Ant script copies all properties (*.properties) files from \WEB-INF\src\java to \WEB-INF\classes. The Ant script also deletes the \WEB-INF\classes directory. Thus, create a log4j.properties file in the \WEB-INF\src\java directory with the following contents:

```
log4j.rootLogger=WARN, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%5p] %d{mm:ss}
(%F:%M:%L) %n%m%n%n
```

The code above sends output to the stdout of the application with the priority, date time stamp, file name, method name, line number and the log message. Logging is turned on for classes under the Struts package and the strutsTutorial code.

To learn more about Log4J, read the online documentation at <http://jakarta.apache.org/log4j/docs/manual.html>, and then read the JavaDoc at <http://jakarta.apache.org/log4j/docs/api/index.html>. Look up the class org.apache.log4j.PatternLayout in the JavaDocs at the top of the file is a list of conversion characters for the output log pattern. You can use the conversion characters to customize what gets output to the log.

Putting log4j on the classpath (copying the jar file to WEB-INF\lib), causes the Commons Logging to use it. The log4J framework finds the log4j.properties file and uses it to create the output logger.

If you start having problems with Struts, then set up the logging level of Struts to debug by adding the following line to log4j.properties file:

```
log4j.logger.org.apache.struts=DEBUG
```

Underneath the covers, we are using Log4J. However, if we want to use logging in our own code, we should use Commons Logging, which allows us to switch to other logging systems if necessary. Thus, we will use the Commons Logging API in our own code. To learn more about Commons Logging, read the “short online manual” at <http://jakarta.apache.org/commons/logging/userguide.html>.

Edit the `UserRegistrationAction` by importing the two Commons Logging classes and putting a trace call in the `execute` method as follows:

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

...
private static Log log = LogFactory.getLog(UserRegistrationAction.class);

public ActionForward execute(...) throws Exception {
    log.trace("In execute method of UserRegistrationAction");
    return mapping.findForward("success");
}
```

You will need to add the Commons Logging Jar file to the `compile.classpath` in `\WEB-INF\src\java\build.xml` file as follows:

```
<path id="compile.classpath">
    <pathelement path="lib/commons-beanutils.jar"/>
    <pathelement path="lib/commons-digester.jar"/>
    <pathelement path="lib/struts.jar"/>
    <pathelement path="classes"/>
    <pathelement path="{classpath}"/>
    <pathelement path="{servlet.jar}"/>
    <pathelement path="lib/commons-logging.jar"/>
</path>
```

Then to get the above trace statement to print out to the log you need to add this line to the `log4j.properties` file:

```
log4j.logger.strutsTutorial=DEBUG
```

Rebuild and deploy the war file, re-enter the url to exercise the action class and look for the log line in the app server's console output.

There are six levels of logging as follows listed in order of importance: fatal, error, warn, info, debug and trace. The log object has the following methods that you can use to log messages.

```
log.fatal(Object message);
log.fatal(Object message, Throwable t);
log.error(Object message);
log.error(Object message, Throwable t);
log.warn(Object message);
log.warn(Object message, Throwable t);
log.info(Object message);
log.info(Object message, Throwable t);
log.debug(Object message);
log.debug(Object message, Throwable t);
log.trace(Object message);
log.trace(Object message, Throwable t);
```

Logging is nearly essential for debugging Struts applications. You must use logging; otherwise, your debugging sessions may be like flying blind.

Write Your First ActionForm

ActionForms represent request data coming from the browser. ActionForms are used to populate HTML forms to display to end users and to collect data from HTML forms. In order to create an ActionForm, you need to follow these steps:

1. Create a new class in the `strutsTutorial` package called `UserRegistrationForm` that subclasses `org.apache.struts.action.ActionForm` as follows:

```
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;

import javax.servlet.http.HttpServletRequest;

public class UserRegistrationForm extends ActionForm {
```

2. Now you need to create JavaBean properties for all the fields that you want to collect from the HTML form. Let's create `firstName`, `lastName`, `userName`, `password`, `passwordCheck` (make sure they entered the right password), `e-mail`, `phone`, `fax` and `registered` (whether or not they are already registered) properties. Add the following fields:

```
private String firstName;
private String lastName;
private String userName;
private String password;
private String passwordCheck;
private String email;
private String phone;
private String fax;
private boolean registered;
```

Add getter and setter methods for each field as follows:

```
public String getEmail() {
    return email;
}
public void setEmail(String string) {
    email = string;
}
```

3. Now you have defined all of the properties for the form. (Reminder: each getter and setter pair defines a property.) Next, you need to override the reset method. The reset method gets called each time a request is made. The reset method allows you to reset the fields to their default value. Here is an example of overwriting the reset method of the ActionForm:

```
public void reset(ActionMapping mapping,
                  HttpServletRequest request) {
    firstName=null;
    lastName=null;
    userName=null;
    password=null;
    passwordCheck=null;
    email=null;
    phone=null;
    fax=null;
    registered=false;
}
```

If you like, please print out a trace method to this method using the logging API, e.g., `log.trace("reset")`.

4. Next you need validate the user entered valid values. In order to do this, you need to override the validate method as follows:

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

    if (firstName==null
        || firstName.trim().equals("")){
        errors.add("firstName",
            new ActionError(
                "userRegistration.firstName.problem"));
    }
    ...
    return errors;
}
```

The validate method returns a list of errors (ActionErrors). The ActionErrors display on the input HTML form. You use your Java programming skills to validate if their typing skills are up to task. The above code checks to see that `firstName` is present; if it is not present (i.e., it is null or blank), then you add an `ActionError` to the `ActionErrors` collection. Notice that when you construct an `ActionError` object, you must pass it a key into the resource bundle (“`userRegistration.firstName`”). Thus, we need to add a value to this key in the Resource bundle.

Please open the file `C:\strutsTutorial\WEB-INF\src\java\resources\application.properties`. Add a key value pair as follows:

```
userRegistration.firstName.problem=The first name was blank
```

If the `firstName` is blank, the control gets redirected back to the input form, and the above message displays. Using a similar technique, validate all the fields.

Write Your First Input View (JSP Page)

Next, we want to create an HTML form in JSP that will act as the input to our Action. The input is like the input view, while the forwards are like output views. In order to create the input view, you will do the following:

1. Create a JSP page called `userRegistration.jsp` in the `c:\strutsTutorial` directory.
2. Import both the Struts HTML and Struts Bean tag libraries. The tag libraries have already been imported into the `web.xml` file as follows:

```
<taglib>
  <taglib-uri>/tags/struts-bean</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld
</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/tags/struts-html</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld
</taglib-location>
</taglib>
```

One of the advantages of using the `blank.war` file is that all the things you need are already configured. You just add the parts that are needed for your application. To import the two tag libraries, you would use the `taglib` directive as follows:

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

3. Use the `html:form` tag to associate the form with the Action. The `html:form` tag associates the form to an action mapping. You use the `action` attribute to specify the path of the action mapping as follows:

```
<html:form action="userRegistration">
```

4. Output the errors associated with this form with the `html:errors` tag. `ActionForms` have a `validate` method that can return `ActionErrors`. Add this to the JSP:

```
<html:errors/>
```

Note there are better ways to do this. Struts 1.1 added `html:messages`, which is nicer as it allows you to get the markup language out of the resource bundle. This is covered in more detail later.

5. Update the Action to associate the Action with the ActionForm and input JSP. In order to do this, you need to edit the `struts-config.xml` file. If you do not feel comfortable editing an XML file, then use the *Struts Console*. Add the following form-bean element under the form-beans element as follows:

```
<form-bean name="userRegistrationForm"
           type="strutsTutorial.UserRegistrationForm" />
```

The code above binds the name `userRegistration` to the form you created earlier: `strutsTutorial.UserRegistrationForm`.

Now that you have added the form-bean element, you need to associate the `userRegistration` action mapping with this form as follows:

```
<action path="/userRegistration"
        type="strutsTutorial.UserRegistrationAction"
        name="userRegistrationForm"
        input="/userRegistration.jsp">
    <forward name="success" path="/regSuccess.jsp" />
</action>
```

Notice the addition of the `name` and `input` attributes. The `name` attribute associates this action mapping with the `userRegistrationForm` ActionForm that you defined earlier. The `input` attribute associates this action mapping with the input JSP. If there are any validation errors, the `execute` method of the action will not get called; instead the control will go back to the `userRegistration.jsp` file until the form has no `ActionErrors` associated with it.

6. Create the labels for the Form fields in the resource bundle. Each field needs to have a label associated with it. Add the following to the resource bundle (`c:/strutsTutorial/WEB-INF/src/java/resources/application.properties`):

```
userRegistration.firstName=First Name
userRegistration.lastName=Last Name
userRegistration.userName=User Name
userRegistration.password=Password
userRegistration.email=Email
userRegistration.phone=Phone
userRegistration.fax=Fax
```

You could instead hard code the values in the JSP page. Putting the value in the resource bundle allows you to internationalize your application.

7. Use the `bean:message` to output the labels. When you want to output labels in the JSP from the resource bundle, you can use the `bean:message` tag. The `bean:message` tag looks up the value in the resource bundle and outputs it from the JSP. The following outputs the label for the `firstName` from the resource bundle:

```
<bean:message key="userRegistration.firstName" />
```

8. Use the `html:text` tag to associate the `ActionForm`'s properties to the HTML form's fields. The `html:text` associates an HTML text field with an `ActionForm` property as follows:

```
<html:text property="firstName" />
```

The above associates the HTML text field with the `firstName` property from your `ActionForm`. The `html:form` tag is associated with the `ActionForm` via the action mapping. The individual text fields are associated with the `ActionForm`'s properties using the `html:text` tag.

9. Create an `html:submit` tag and an `html:cancel` tag to render a submit button and a cancel button in html as follows:

```
<html:submit />  
...  
<html:cancel />
```

At this point you should be able to deploy and test your Struts application. The Action has not been wired to do much of anything yet. But the form will submit to the Action. And, if a form field is invalid the control will be forwarded back to the input form. Try this out by leaving the `firstName` field blank.

If you are having problems, you may want to compare what you have written to the solution. Here is what the `userRegistration.jsp` looks like after you finish (your HTML may look different):

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

  <head>
    <title>User Registration</title>
  </head>

  <body>
    <h1>User Registration</h1>

<html:errors/>

    <table>
    <html:form action="userRegistration">
      <tr>
        <td>
          <bean:message key="userRegistration.firstName" />*
        </td>
        <td>
          <html:text property="firstName" />
        </td>
      </tr>
      <tr>
        <td>
          <bean:message key="userRegistration.lastName" />*
        </td>
        <td>
          <html:text property="lastName" />
        </td>
      </tr>
      <tr>
        <td>
          <bean:message key="userRegistration.userName" />*
        </td>
        <td>
          <html:text property="userName" />
        </td>
      </tr>
      <tr>
        <td>
          <bean:message key="userRegistration.email" />*
        </td>
        <td>
          <html:text property="email" />
        </td>
      </tr>
    </tr>
  </table>
```

```
<td>
  <bean:message key="userRegistration.phone" />
</td>
<td>
  <html:text property="phone" />
</td>
</tr>
<tr>
<td>
  <bean:message key="userRegistration.fax" />
</td>
<td>
  <html:text property="fax" />
</td>
</tr>
<tr>
<td>
  <bean:message key="userRegistration.password" />*
</td>
<td>
  <html:password property="password" />
</td>
</tr>
<tr>
<td>
  <bean:message key="userRegistration.password" />*
</td>
<td>
  <html:password property="passwordCheck" />
</td>
</tr>
<tr>
<td>
  <html:submit />
</td>
<td>
  <html:cancel />
</td>
</tr>

</html:form>
</table>
</body>
</html>
```

The form should look like this when it first gets loaded. (You load the form by going to <http://localhost:8080/strutsTutorial/userRegistration.jsp>.)

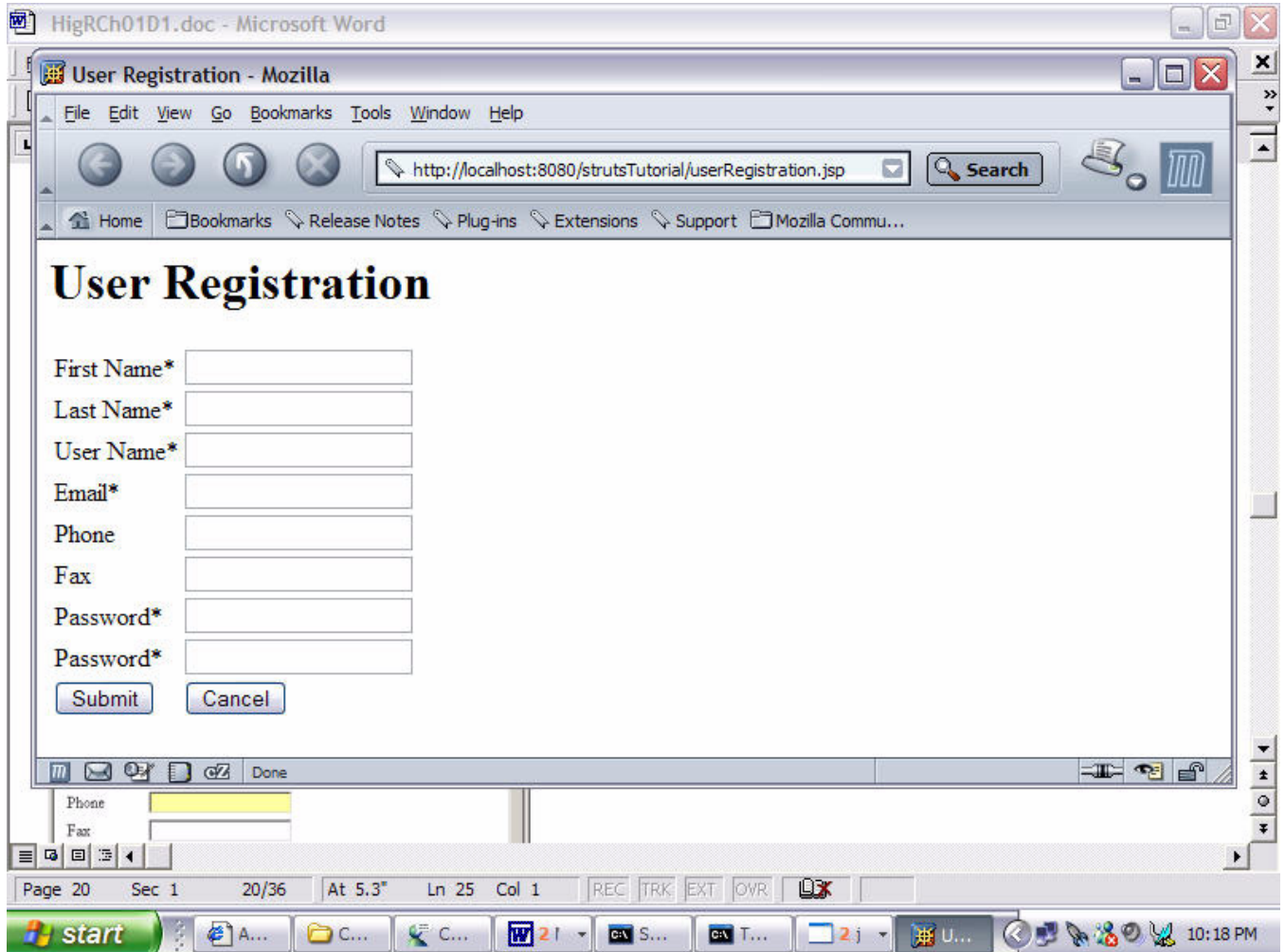


Figure 1.4 User Registration JSP

If you leave the `firstName` blank, then you should get a form that looks like this.



Figure 1.5 User Registration JSP with validation errors

Notice that the error message associated with the `firstName` displays, since the `firstName` was left blank. It is instructive to view the logs as you run the example to see the underlying interactions of the Struts framework. Once you complete the form and hit the Submit button, the `execute` method of `getUserRegistrationAction` is invoked. Currently the `execute` method just forwards to `regSuccess.jsp`, which is mapped into the success forward, whether or not the Cancel button is pressed.

Update the Action to Handle the Form and Cancel Buttons

Let's do something with the `ActionForm` that gets passed to the Action. Once you fill in the form correctly (no validation errors) and hit the submit button, the `execute` method of the `UserRegistrationAction` is invoked. Actually, the `execute` method gets invoked whether or not you hit the submit button or the cancel button.

You need check to see if the cancel button was clicked; it was clicked forward to welcome. The welcome forward was setup by the authors of `blank.war`, and it forwards to `"/Welcome.do"`, which forwards to `/pages/Welcome.jsp`. Check out the `struts-config.xml` file to figure out how they did this. To check and see if the cancel button was clicked, you need to use the `isCancelled` method of the Action class in the `execute` method as follows:

```
public ActionForward execute(...) {...  
    ...  
    if (isCancelled(request)) {  
        log.debug("Cancel Button was pushed!");  
        return mapping.findForward("welcome");  
    }  
    ...  
}
```

The `isCancelled` method takes an `HttpServletRequest` as an argument. The `execute` method was passed an `HttpServletRequest`.

Next, you need to cast the `ActionForm` to an `UserRegistrationForm`. In order to use the form that was submitted to the action, you need to cast the `ActionForm` to the proper type. Thus, you will need to cast the `ActionForm` that was passed to the `execute` method to a `UserRegistrationForm` as follows:

```
UserRegistrationForm userForm =  
    (UserRegistrationForm) form;
```

Now you can start using the `UserRegistrationForm` like this:

```
log.debug("userForm firstName" + userForm.getFirstName());
```

For now, just print out the `firstName` with the logging utility. In the next section, you'll do something more useful with this form—you will write it to a database.

Set up the Database Pooling with Struts

This is an optional section. You can skip this section and continue the tutorial. However, if you want to use Struts DataSources, then skipping this session is not an option.

Struts DataSources allow you to get the benefits of data sources without being tied to any one vendor's implementation. On one hand, it can make your application more portable as configuring J2EE datasources is vendor specific. On the other hand, you cannot use some of the more advanced features of your specific vendor's implementation. If you are not sure if you are going to use Struts DataSources, then follow along and make your decision after this section.

You need both commons-pooling and commons-logging to get Struts Datasources to work, but this is not mentioned in the online documentation, and the required jar files do not ship with Blank.war or with Struts 1.1 at all. In fact, Struts 1.1., requires the Struts Legacy jar file. Note that if you are using Tomcat5, then both commons-pooling and commons-logging ship in the common/lib folder. If you are using another application server with Struts Datasources, you need to download them.

Currently neither http://jakarta.apache.org/struts/userGuide/configuration.html#data-source_config nor <http://jakarta.apache.org/struts/faqs/database.html> make note of this.

Struts Datasources require commons-pooling and commons-logging. The blank.war does not have commons-pooling or commons-logging. If you want to use Struts datasource, you will need to download commons-pooling and commons-logging from the following locations:

- <http://jakarta.apache.org/site/binindex.cgi#commons-logging>
- <http://jakarta.apache.org/site/binindex.cgi#commons-pool>

Download the above archives, and extract them. Then, copy the jar files commons-pool-1.1.jar, and commons-logging-1.1.jar from the archives into the WEB-INF/lib directory of your web application. Note that Tomcat 5 ships with commons-logging and commons-pool so you do not need to download and install commons-logging and commons-pool if you are using Tomcat 5.

In addition to the above two jar files, you will need to use the struts-legacy.jar jar file that ships with Struts 1.1. Copy the struts-legacy.jar file (C:\tools\jakarta-struts-1.1\lib\struts-legacy.jar) to the WEB-INF/lib directory of your web application.

This might be a moot point depending on how soon Struts 1.2 comes out and if commons-pooling and commons-logging ship with Struts 1.2.

You need to create a table in your database of choice, similar to the one whose DDL is listed below:

```
CREATE TABLE USER(  
    EMAIL VARCHAR(80),  
    FIRST_NAME VARCHAR(80),  
    LAST_NAME VARCHAR(80),  
    PASSWORD VARCHAR(11),  
    PHONE VARCHAR(11),  
    FAX VARCHAR(11),  
    CONSTRAINT USER_PK PRIMARY KEY (EMAIL)  
);
```

You need to obtain a JDBC driver for your database of choice and obtain the JDBC URL. Then using the JDBC driver name and the JDBC URL, write the following in your Struts-Config.xml file (before the formbeans element).

```
<data-sources>  
  <data-source  
    type="org.apache.commons.dbcp.BasicDataSource"  
    key="userDB">  
  
    <set-property property="driverClassName"  
      value="org.hsqldb.jdbcDriver" />  
    <set-property property="url"  
      value="jdbc:hsqldb:c:/strutsTutorial/db" />  
    <set-property property="username" value="sa" />  
    <set-property property="password" value="" />  
  </data-source>  
</data-sources>
```

Notice that the set-property sets a property called driverClassName, which is the class name of your JDBC driver, and set-property sets the url, which is the JDBC URL, of the database that has your new table. Also, note that you will need to specify the username and password for the database using set-property.

You will need to copy the jar file that contains the JDBC driver onto the classpath somehow. Most application servers have a folder that contains files that are shared by all web applications; you can put the JDBC driver jar file there.

Note I added an Ant target called `builddb` that creates the table with DDL using the ant task `sql`. Thus, I added the following to the Ant script to create the database table. You can use a similar technique or use the tools that ship with your database:

```
<target name="builddb"
  description="builds the database tables">
  <sql driver="org.hsqldb.jdbcDriver"
    userid="sa"
    password=""
    url="jdbc:hsqldb:c:/strutsTutorial/db">
    <classpath>
      <pathelement
        path="/tools/hsqldb/lib/hsqldb.jar"/>
    </classpath>

CREATE TABLE USER(
  EMAIL VARCHAR(80),
  FIRST_NAME VARCHAR(80),
  LAST_NAME VARCHAR(80),
  PASSWORD VARCHAR(11),
  PHONE VARCHAR(11),
  FAX VARCHAR(11),
  CONSTRAINT USER_PK PRIMARY KEY
                        (EMAIL)
);

  </sql>
</target>
```

Note that I used HSQL DB as my database engine, which can be found at <http://hsqldb.sourceforge.net/>. You should be able to use any database that has a suitable JDBC driver.

Once you have the database installed and configured, you can start accessing it inside of an Action as follows:

```
import java.sql.Connection;
import javax.sql.DataSource;
import java.sql.PreparedStatement;

...
public ActionForward execute(...)...{
    DataSource dataSource =
        getDataSource(request, "userDB");

    Connection conn = dataSource.getConnection();
```

Warning! At this point, I strongly recommend that you never put database access code inside of an Action's execute method, except in a tutorial. This should almost always be delegated to a Data Access Object. However, for the sake of simplicity, go ahead and insert the UserRegistrationForm into that database inside of the execute method as follows:

```
UserRegistrationForm userForm = (UserRegistrationForm) form;

log.debug("userForm firstName" + userForm.getFirstName());

DataSource dataSource = getDataSource(request, "userDB");
Connection conn = dataSource.getConnection();

try{
    PreparedStatement statement = conn.prepareStatement(
        "insert into USER " +
        "(EMAIL, FIRST_NAME, LAST_NAME, PASSWORD, PHONE, FAX)" +
        " values (?, ?, ?, ?, ?, ?)");

    statement.setString(1, userForm.getEmail());
    statement.setString(2, userForm.getFirstName());
    statement.setString(3, userForm.getLastName());
    statement.setString(4, userForm.getPassword());
    statement.setString(5, userForm.getPhone());
    statement.setString(6, userForm.getFax());
    statement.executeUpdate();
}finally{
    conn.close();
}

return mapping.findForward("success");
```

Now, test and deploy the application. If you have gotten this far and things are working, then you have made some good progress. So far you have created an Action, ActionForm, ActionMapping and set up a Struts DataSource.

Exception Handling with Struts

Bad things happen to good programs. It is our job as fearless Struts programmers to prevent these bad things from showing up to the end user. You probably do not want the end user of your system to see a Stack Trace. An end user seeing a Stack Trace is like any computer user seeing the “Blue Screen of Death” (generally not a very pleasant experience for anyone).

It just so happens that when you enter an e-mail address into two User Registrations, you get a nasty error message as the e-mail address is the primary key of the database table. Now, one could argue that this is not a true “exceptional” condition, as it can happen during the normal use of the application, but this is not a tutorial on design issues. This is a tutorial on Struts, and this situation gives us an excellent opportunity to explain Struts declarative exception handling.

If you enter in the same e-mail address twice into two User Registrations, the system will throw a `java.sql.SQLException`. In Struts, you can set up an exception handler to handle an exception.

An exception handler allows you to declaratively handle an exception in the `struts-config.xml` file by associating an exception to a user friendly message and a user friendly JSP page that will display if the exception occurs.

Let’s set up an exception handler for this situation. Follow these steps:

1. Create a JSP file called `userRegistrationException.jsp` in the root directory of the project (`c:\strutsTutorial`).

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

<html>

<head>
<title>
User Registration Had Some Problems
</title>
</head>

<body>
<h1>User Registration Had Some Problems!</h1>
<html:errors/>
</body>

</html>
```

Notice the use of `html:errors` to display the error message associated with the exception.

2. Add an entry in the resource bundle under the key `userRegistration.sql.exception` that explains the nature of the problem in terms that the end user understands. This message will be used by the exception handler. Specifically, you can display this message using the `html:errors` tag in the `userRegistrationException.jsp` file. Edit the properties file associated with the resource bundle (located at `C:\strutsTutorial\WEB-INF\src\java\resources\application.properties` if you have been following along with the home game version of the Struts tutorial).

```
userRegistration.sql.exception=There was a problem adding the User. \n The
most likely problem is the user already exists or the email\n address is
being used by another user.
```

(The code above is all one line.)

3. Add an exception handler in the action mapping for `/userRegistration` that handles `java.sql.SQLException` as follows:

```
<action path="/userRegistration"
        type="strutsTutorial.UserRegistrationAction"
        name="userRegistrationForm"
        input="/userRegistration.jsp">
  <exception type="java.sql.SQLException"
            key="userRegistration.sql.exception"
            path="/userRegistrationException.jsp" />

  <forward name="success" path="/regSuccess.jsp" />
  <forward name="failure" path="/regFailure.jsp" />
</action>
```

Notice that you add the exception handler by using the exception element (highlighted above). The above exception element has three attributes: `type`, `key` and `path`. The `type` attribute associates this exception handler with the exception `java.sql.SQLException`. The `key` attribute associates the exception handler with a user friendly message out of the resource bundle. The `path` attribute associates the exception handler with the page that will display if the exception occurs.

If you do everything right, you get the following when the exception occurs.

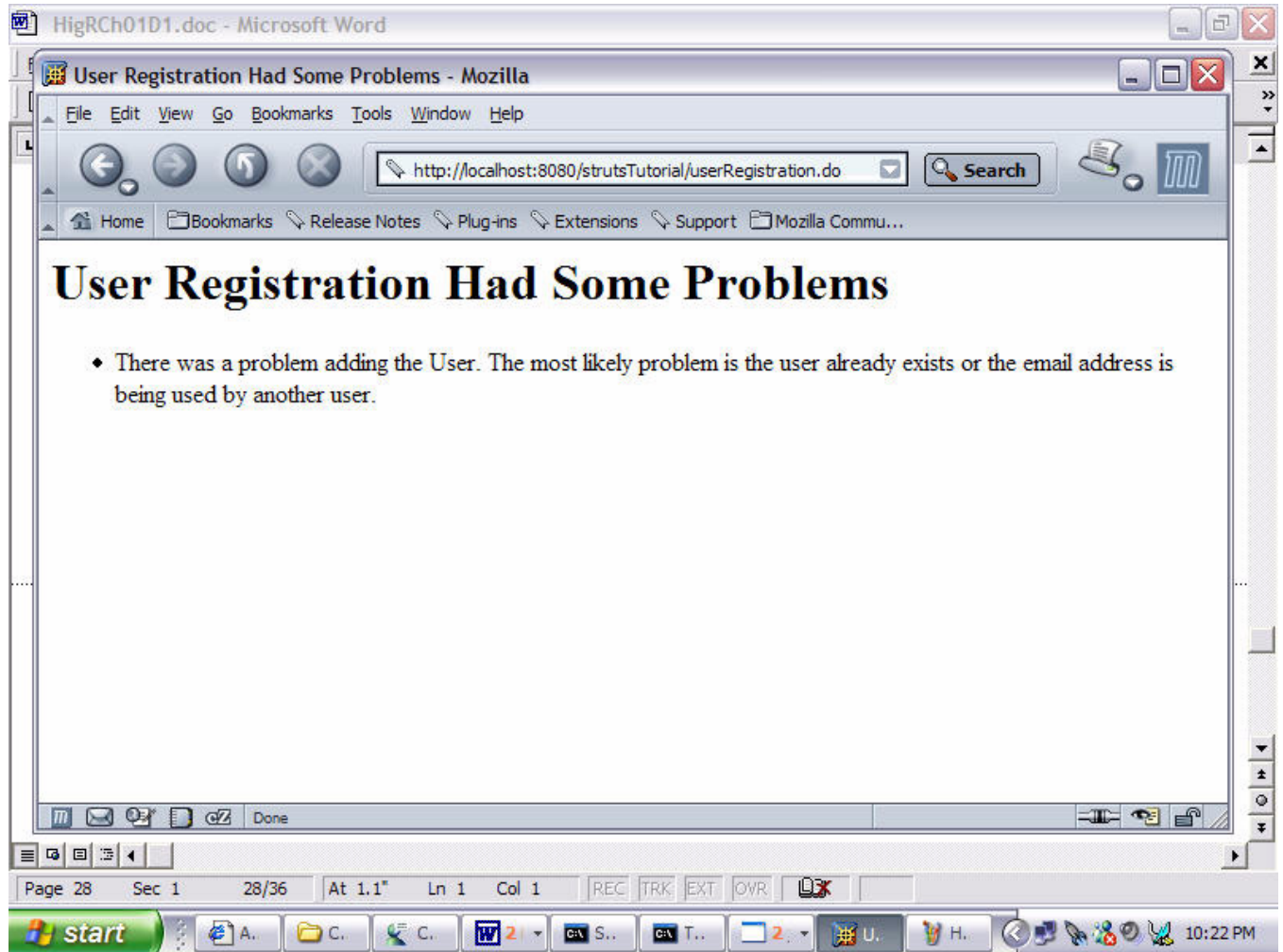


Figure 1.6 Declarative Exception Handling

Display an Object with Struts Tags

Struts supports a Model 2 architecture. The Actions interact with the model and perform control flow operations, like which view is the next view to display. Then, Actions delegate to JSP (or other technologies) to display objects from the model.

To start using Struts with this tutorial, follow these steps:

1. Add an attribute called `attribute` to the mapping that causes the ActionForm to be mapped into scope as follows:

```
<action path="/userRegistration"
        type="strutsTutorial.UserRegistrationAction"
        name="userRegistrationForm"
        attribute="user"
        input="/userRegistration.jsp">
    ...
```

Notice the above action mapping uses the attribute called `attribute`. The attribute maps the ActionForm into a scope (session scope by default) under “user”. Now that the ActionForm is in session scope, you can display properties from the ActionForm in the view.

2. Edit the `regSuccess.jsp` that you created earlier. The `regSuccess.jsp` is an ActionForward for the UserRegistrationAction. The `regSuccess.jsp` is the output view for the Action. In order to display the ActionForm, you could use the Struts bean tag library.
3. Import the bean tag library into the JSP as follows:

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

4. Use the `bean:write` to output properties of the ActionForm

```
<bean:write name="user" property="firstName" />
```

The code above prints out the `firstName` property of the user object. Use the above technique to print out all of the properties of the user object.

When you are done with the JSP, it should look something like this:

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

<html>

<head>
<title>
User Registration Was Successful!
```

```
</title>
</head>

<body>
<h1>User Registration Was Successful!</h1>
</body>

<table>
  <tr>

    <td>
      <bean:message key="userRegistration.firstName" />
    </td>

    <td>
      <bean:write name="user" property="firstName" />
    </td>

  </tr>

  <tr>

    <td>
      <bean:message key="userRegistration.lastName" />
    </td>

    <td>
      <bean:write name="user" property="lastName" />
    </td>

  </tr>

  <tr>

    <td>
      <bean:message key="userRegistration.email" />
    </td>

    <td>
      <bean:write name="user" property="email" />
    </td>

  </tr>
</table>

</html>
```

Using Logic Tags to Iterate over Users

Struts provides logic tags that enable you to have display logic in your view without putting Java code in your JSP with Java scriptlets. To start using the Logic tags, follow these steps.

1. Create a JavaBean class called User to hold a user with email, firstName and lastName properties. Here is a possible implementation (partial listing):

```
package strutsTutorial;

import java.io.Serializable;

public class User implements Serializable {

    private String lastName;
    private String firstName;
    private String email;

    public String getEmail() {
        return email;
    }

    ...

    public void setEmail(String string) {
        email = string;
    }

    ...
}
```

2. Create a new Action called DisplayAllUsersAction.

```
public class DisplayAllUsersAction extends Action {
```

In the new Action's execute method, complete the following steps:

3. Get the userDB datasource.

```
DataSource dataSource = getDataSource(request, "userDB");
```

4. Create a DB connection using the datasource.

```
Connection conn = dataSource.getConnection();
Statement statement = conn.createStatement();
```

5. Query the DB, and copy the results into a collection of the `User` JavaBean:

```
ResultSet rs =
statement.executeQuery("select FIRST_NAME, LAST_NAME, EMAIL from USER");

List list = new ArrayList(50);

while (rs.next()){
    String firstName = rs.getString(1);
    String lastName = rs.getString(2);
    String email = rs.getString(3);

    User user = new User();
    user.setEmail(email);
    user.setFirstName(firstName);
    user.setLastName(lastName);
    list.add(user);
}

if (list.size() > 0){
    request.setAttribute("users", list);
}
```

Tip: Don't forget to close the connection using a try/finally block.

Warning! You do not typically put SQL statements and JDBC code directly in an Action. This type of code should be in a `DataAccessObject`. A `DataAccessObject` would encapsulate the CRUD access for a particular domain object. The `DataAccessObject` is part of the Model of the application.

6. Create a new JSP called `userRegistrationList.jsp`.

In the new JSP, perform the following steps:

7. Import the logic tag library into the `userRegistrationList.jsp`.

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

8. Check to see if the users are in scope with the `logic:present` tag.

```
<logic:present name="users">
    ... (Step 9 goes here)
</logic:present>
```

9. If the users are in scope, iterate through them.

```
<logic:iterate id="user" name="users">
    ... (Step 10 goes here)
</logic:iterate>
```

10. For each iteration, print out the firstName, lastName and email using bean:write

```
<bean:write name="user"
           property="firstName"/>

<bean:write name="user"
           property="lastName"/>

<bean:write name="user"
           property="email"/>
```

One possible implementation for the JSP is as follows:

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

<html>

  <head>
    <title>User Registration List</title>
  </head>

  <body>
    <h1>User Registration List</h1>

    <logic:present name="users">

      <table border="1">
        <tr>
          <th>
            <bean:message
              key="userRegistration.firstName"/>
          </th>

          <th>
            <bean:message
              key="userRegistration.lastName" />
          </th>

          <th>
            <bean:message
              key="userRegistration.email" />
          </th>
        </tr>
        <logic:iterate id="user" name="users">
          <tr>

            <td>
              <bean:write name="user"
```

```
                property="firstName"/>
            </td>

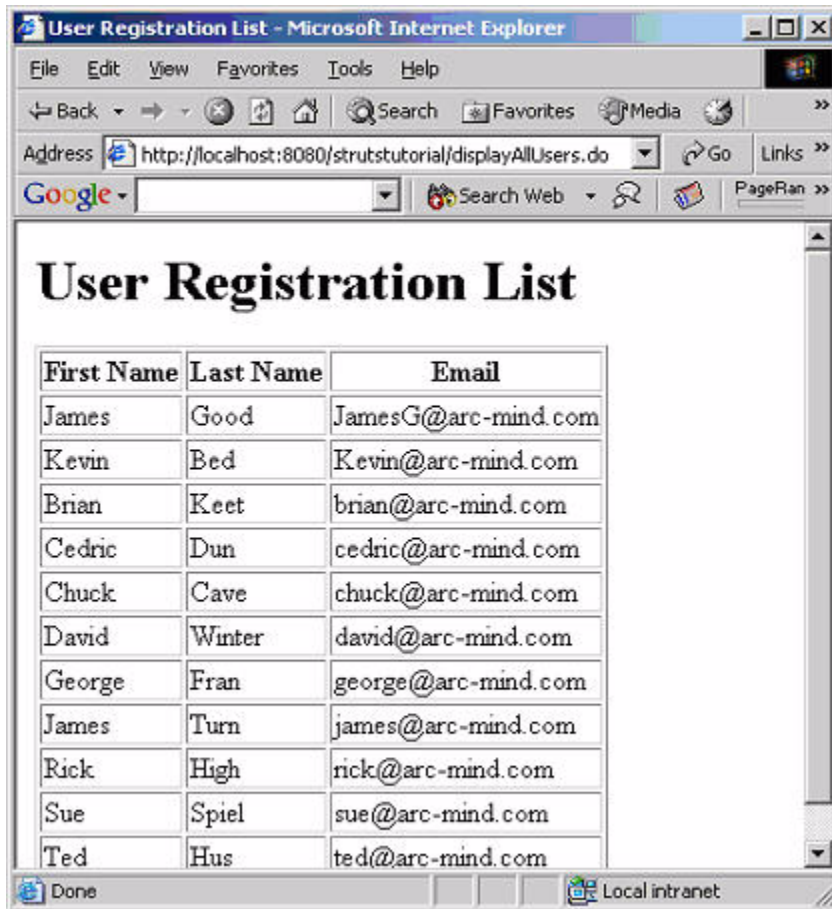
            <td>
                <bean:write name="user"
                    property="lastName"/>
            </td>

            <td>
                <bean:write name="user"
                    property="email"/>
            </td>

        </tr>

    </logic:iterate>
</table>

</logic:present>
</body>
</html>
```



First Name	Last Name	Email
James	Good	JamesG@arc-mind.com
Kevin	Bed	Kevin@arc-mind.com
Brian	Keet	brian@arc-mind.com
Cedric	Dun	cedric@arc-mind.com
Chuck	Cave	chuck@arc-mind.com
David	Winter	david@arc-mind.com
George	Fran	george@arc-mind.com
James	Turn	james@arc-mind.com
Rick	High	rick@arc-mind.com
Sue	Spiel	sue@arc-mind.com
Ted	Hus	ted@arc-mind.com

Figure 1.7 User Listing

11. Create a new entry in the struts-config.xml file for this new Action.

```
<action path="/displayAllUsers"
        type="strutsTutorial.DisplayAllUsersAction">
    <forward name="success"
            path="/userRegistrationList.jsp"/>
</action>
```

Now you can deploy and test this new Action by going to: <http://localhost:8080/strutstutorial/displayAllUsers.do>

Adjust your domain and port number accordingly.

Summary

Now that you have completed this chapter, you have experienced many different aspects of Struts. While not going into great detail on any one topic, you should have a better idea about the breadth of Struts. The chapters that follow will expand in detail on the breadth and depth of the Struts framework.