

JSF and Struts Classic Reference Manual

Version: 1.0.0.GA

Copyright © 2007 Red Hat

Table of Contents

1. Introduction	1
2. Getting Started Guide for Creating a JSF Application	2
2.1. Creating a Simple JSF Application	2
2.2. Setting Up the Project	2
2.3. The JSF Application Configuration File	2
2.4. Adding Navigation to the Application	4
2.4.1. Adding Two Views (JSP Pages)	4
2.4.1.1. Creating the Transition (Navigation Rule)	5
2.5. Adding a Managed Bean to the Application	6
2.6. Editing the JSP View Files	7
2.6.1. inputname.jsp	7
2.6.2. greeting.jsp	11
2.7. Creating the Start Page	12
2.8. Running the Application	12
2.9. Other relevant resources on the topic	13
3. Getting Started Guide for Creating a Struts Application	14
3.1. Starting Up	14
3.2. Creating the Application Components	15
3.2.1. Creating JSP Page Placeholders	15
3.2.1.1. Creating the Page Placeholders	15
3.2.1.2. Placing the Page Placeholders	15
3.2.2. Creating an Action Mappings	16
3.2.3. Creating a Link	16
3.2.4. Creating a Forward	17
3.2.5. Creating a Global Forward	17
3.2.6. Creating a Form Bean	18
3.3. Generating Stub Coding	18
3.4. Coding the Various Files	19
3.4.1. Java Stub Classes	19
3.4.1.1. GetNameForm.java	19
3.4.1.2. GreetingAction.java	20
3.4.2. JSP Pages	21
3.4.2.1. inputname.jsp	21
3.4.2.2. greeting.jsp	24
3.4.2.3. index.jsp	26
3.5. Compiling the Classes	27
3.6. Running the Application	27
3.7. Other relevant resources on the topic	27
4. Getting Started Struts Validation Examples	29
4.1. Starting Point	29
4.2. Defining the Validation Rule	29
4.3. Client-Side Validation	31
4.4. Server Side Validation	33
4.5. Editing the JSP File	33

4.6. Editing the Action	34
4.7. Editing the Form Bean	34
4.8. Other Resources	35

1

Introduction

The following chapters describe how to deal with classic/old style of JSF and Struts development. We recommend users to use JBoss Seam [http://www.redhat.com/developers/jbds/Getting_Started/GetStartSeamGen.html] to simplify development, but until then you can read about classical JSF and Struts usage here.

Getting Started Guide for Creating a JSF Application

2.1. Creating a Simple JSF Application

We are going to show you how to create a simple JSF application using the JBoss Developer Studio plug-in for Eclipse. The completed application will ask a user to enter a name and click a button. The resulting new page will display the familiar message, "Hello <name>!" This document will show you how to create such an application from the beginning, along the way demonstrating some of the powerful features of JBoss Developer Studio. You will design the JSF application and then run the application from inside JBoss Developer Studio. We'll assume that you have already launched Eclipse with JBoss Developer Studio installed and also that the JBoss Developer Studio perspective is the current one. (If not, make it active by selecting *Window > Open Perspective > Web Development* from the menu bar or by selecting *Window > Open Perspective > Other...* from the menu bar and then selecting *Web Development* from the Select Perspective dialog box.)

2.2. Setting Up the Project

We are first going to create a new project for the application.

- Go to the menu bar and select *File > New > Project...*
- Select *JBoss Tools Web > JSF > JSF Project* in the New Project dialog box
- Click *Next*
- Enter "jsfHello" as the project name.
- Leave everything else as is, and click *Finish*

2.3. The JSF Application Configuration File

A jsfHello node should appear in the upper-left Package Explorer view.

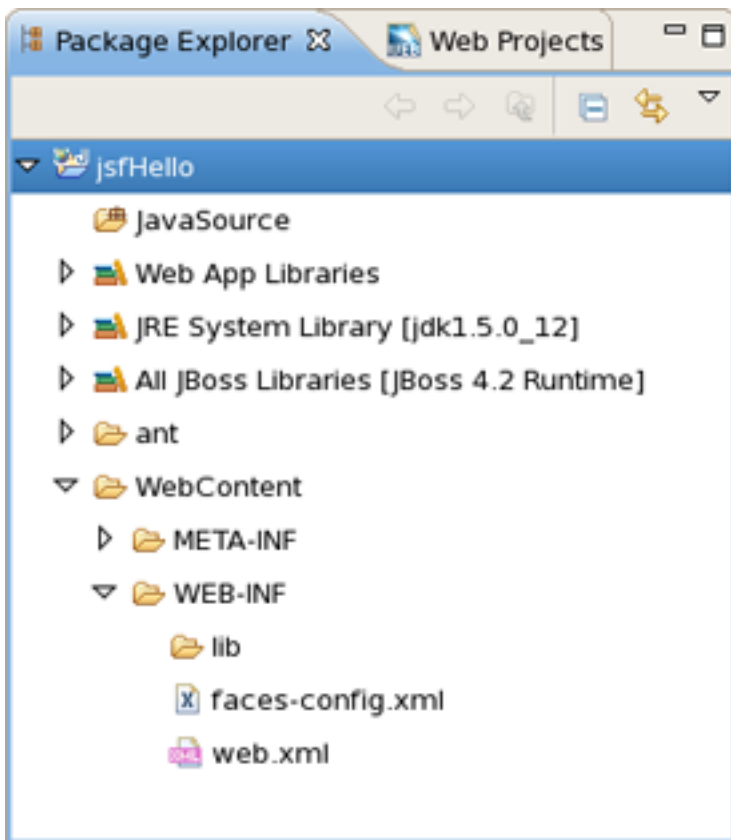


Figure 2.1. Package Explorer View

- Click the plus sign next to *jsfHello* to reveal the child nodes
- Click the plus sign next to *WebContent* under *jsfHello*
- Click the plus sign next to *WEB-INF* under *WebContent*
- Then double-click on the *faces-config.xml* node to display the JSF application configuration file editor

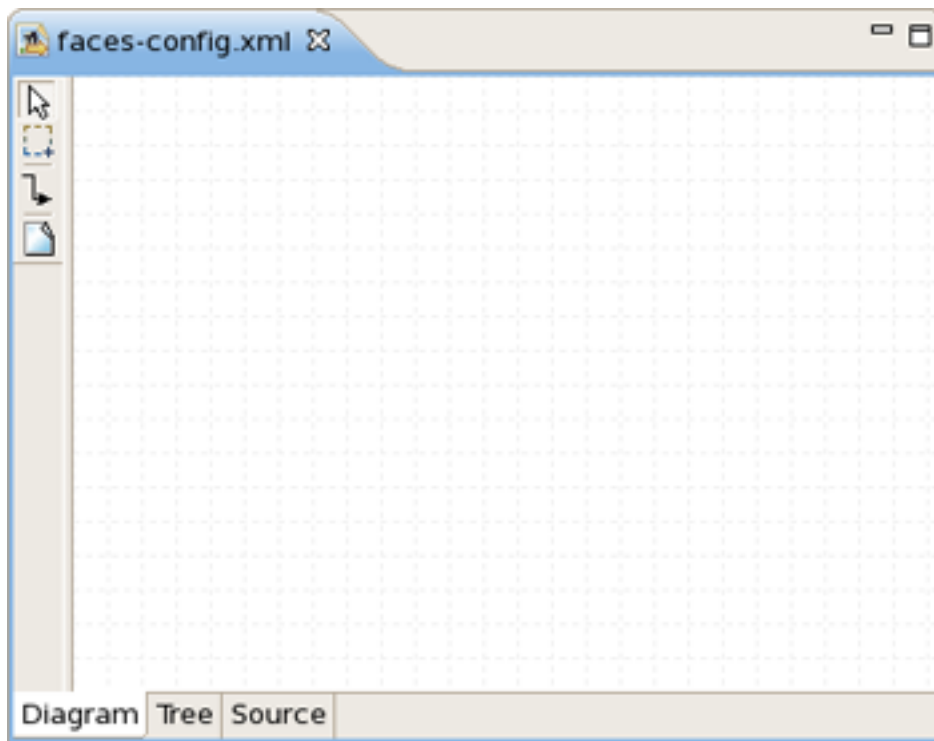


Figure 2.2. Configuration File Editor

2.4. Adding Navigation to the Application

In our simple application, the flow is defined as a single navigation rule connecting two views (presentation files). At this point, we will create the placeholders for the two JSP presentation files and then the navigation rule to connect them as views. Later, we will complete the coding for the JSP presentation files. With JBoss Developer Studio, we can do all of this in the Diagram mode of the configuration file editor.

2.4.1. Adding Two Views (JSP Pages)

- Right-click anywhere on the diagram and select *New View...* from the pop-up menu
- In the dialog box, type *pages/inputname* as the value for From-view-id
- Leave everything else as is
- Click *Finish*

If you look in the Package Explorer view you should see a *pages* folder under WebContent. Opening it will reveal the JSP file you just created

- Back on the diagram, right-click anywhere and select *New View...* from the pop-up menu
- In the dialog box, type *pages/greeting* as the value for From-view-id

- Leave everything else as is
- Click *Finish*

2.4.1.1. Creating the Transition (Navigation Rule)

- In the diagram, select the connection icon third from the top along the upper left side of the diagram



Figure 2.3. Connection Icon

to get an arrow cursor with a two-pronged plug at the arrow's bottom.

- Click on the *pages/inputname* page icon and then click on the *pages/greeting* page icon

A transition should appear between the two icons.

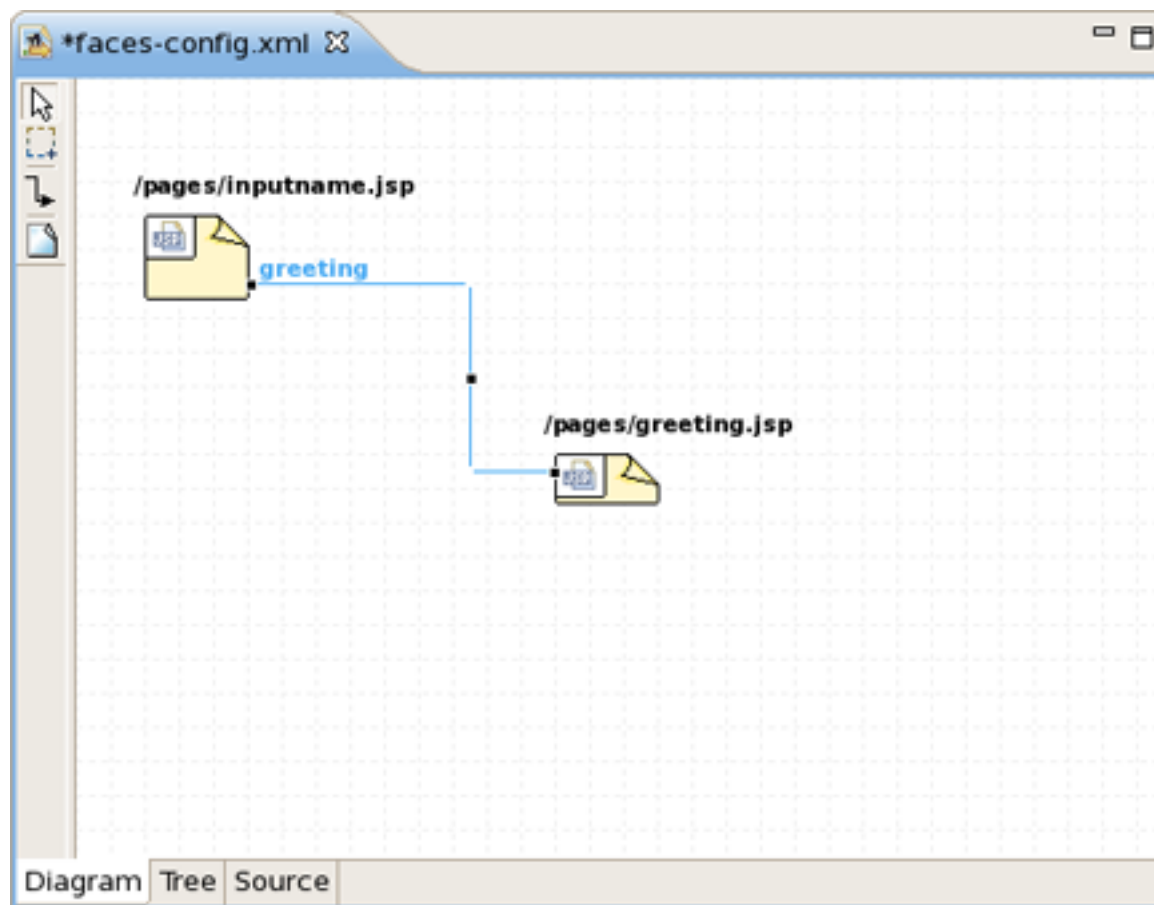


Figure 2.4. Transition Between Two Icons

- Select *File > Save* from the menu bar

2.5. Adding a Managed Bean to the Application

To store data in the application, we will use a managed bean.

- Click on the *Tree* tab at the bottom of the editing window
- Select the *Managed Beans* node and then click the *Add...* button displayed along the right side of the editor window
- Type in *jsfHello.PersonBean* for Class and *personBean* for Name. Leave Scope as is and Generate Source Code as is (checked)
- Click *Finish*
- *personBean* will now be selected and three sections of information: *Managed Bean*, *Properties*, and *Advanced*, will be displayed about it. Under the *Properties* section, click the *Add...* button
- Type in *name* for Property-Name. Leave everything else as is. (When Property- Class is not filled in, String is the assumed type)
- Click *Finish*
- Select the *personBean* node in the tree

You should see this now:

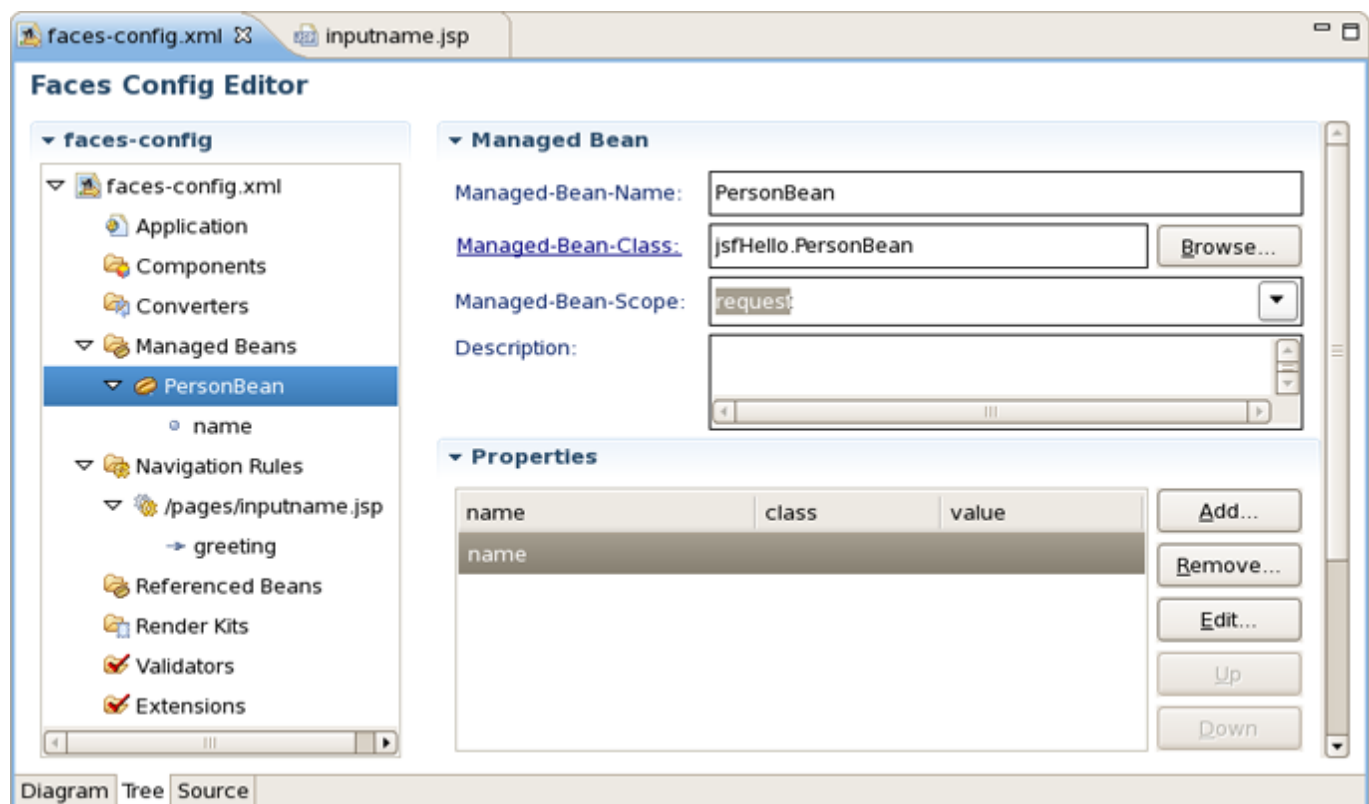


Figure 2.5. Tree View in Config Editor

- Select *File > Save* from the menu bar

You have now registered the *managed bean* and created a *stub-coded class* file for it.

2.6. Editing the JSP View Files

Now we will finish editing the JSP files for our two "views" using JSP Visual Page.

2.6.1. inputname.jsp

- Click on the *Diagram* tab for the configuration file editor
- Open the editor for this first JSP file by double-clicking on the */pages/inputname.jsp* icon

The Visual Page Editor will open in a screen split between source code along the top and a WYSIWIG view along the bottom:

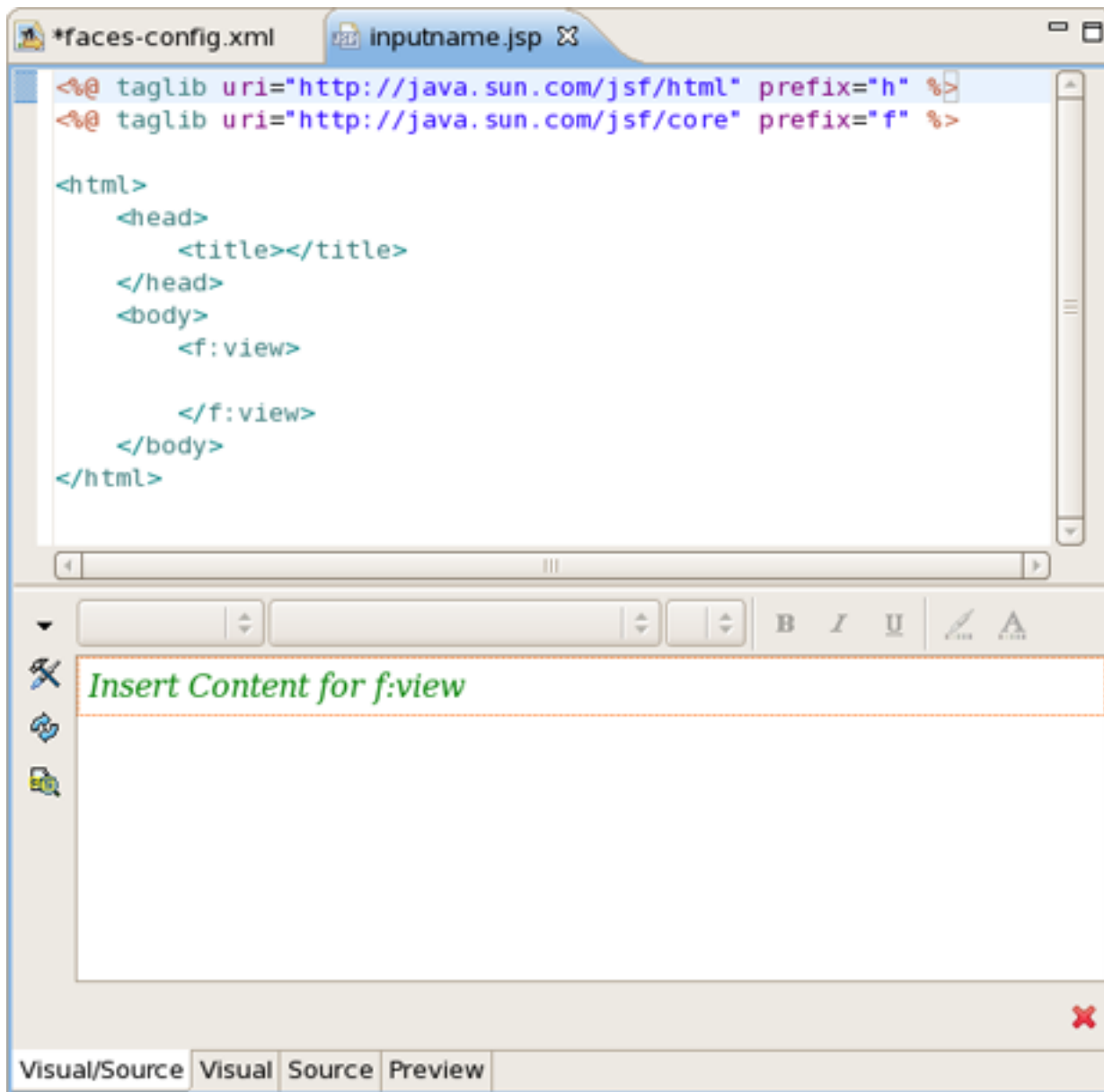


Figure 2.6. Visual Page Editor

Some JSF code is already in the file, because we have chosen a template to create a page.

- Select the *Visual* tab, so we can work with the editor completely in its WYSIWYG mode
- To the right of the editor, in the JBoss Tools Palette, expand the *JSF HTML* palette folder by selecting it

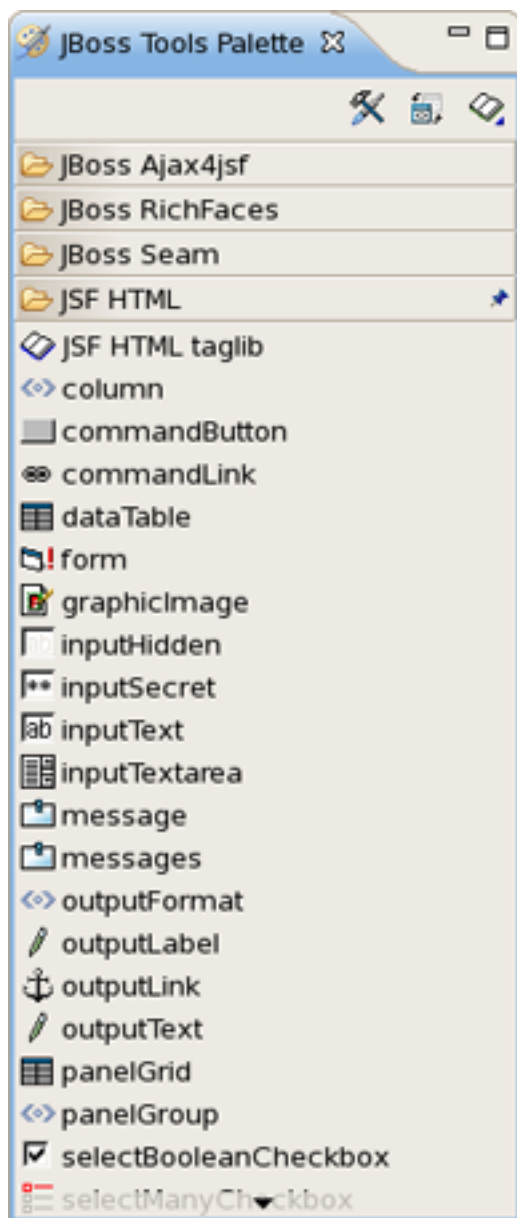


Figure 2.7. JBoss Tools Palette

- Click on *form* within this folder, drag the cursor over to the editor, and drop it inside the red box in the editor
- Another red box will appear inside the first red box
- Right-click on the innermost box and select **<h:form>** Attributes from the menu
- In the value field next to id, type *greeting* and click on the *Close* button
- Type "Please enter name:" inside the boxes
- Select *inputText* within the JSF HTML palette folder and drag it into the innermost box in the editor after "Please enter name:"

- In the attributes dialog, click in the *value* field next to the value attribute and click on the ... button
- Then, select the *Managed Beans > personBean > name* node and click on the *Ok* button
- Back in the attributes dialog, select the *Advanced* tab, type in *name* as the value for the "*id*" attribute, and then click on the *Finish* button
- Select *commandButton* within the JSF HTML palette folder and drag it into the innermost box in the editor after the input box
- In the attributes dialog, click in the value field next to the "*action*" attribute and click on the ... button
- Then, select the *View Actions > greeting* node and click on the *OK* button
- Back in the attributes dialog box, type in "Say Hello" as the value for the value attribute ("Say Hello") and then click on the *Finish* button

The source coding should be something like this now:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
<h:form id="greeting">
<para>Please enter a name:</para>
<h:inputText id="name" value="#{personBean.name}"/>
<h:commandButton value=" Say Hello " action="greeting"/>
</h:form>
</f:view>
</body>
</html>
```

The editor should look like this:

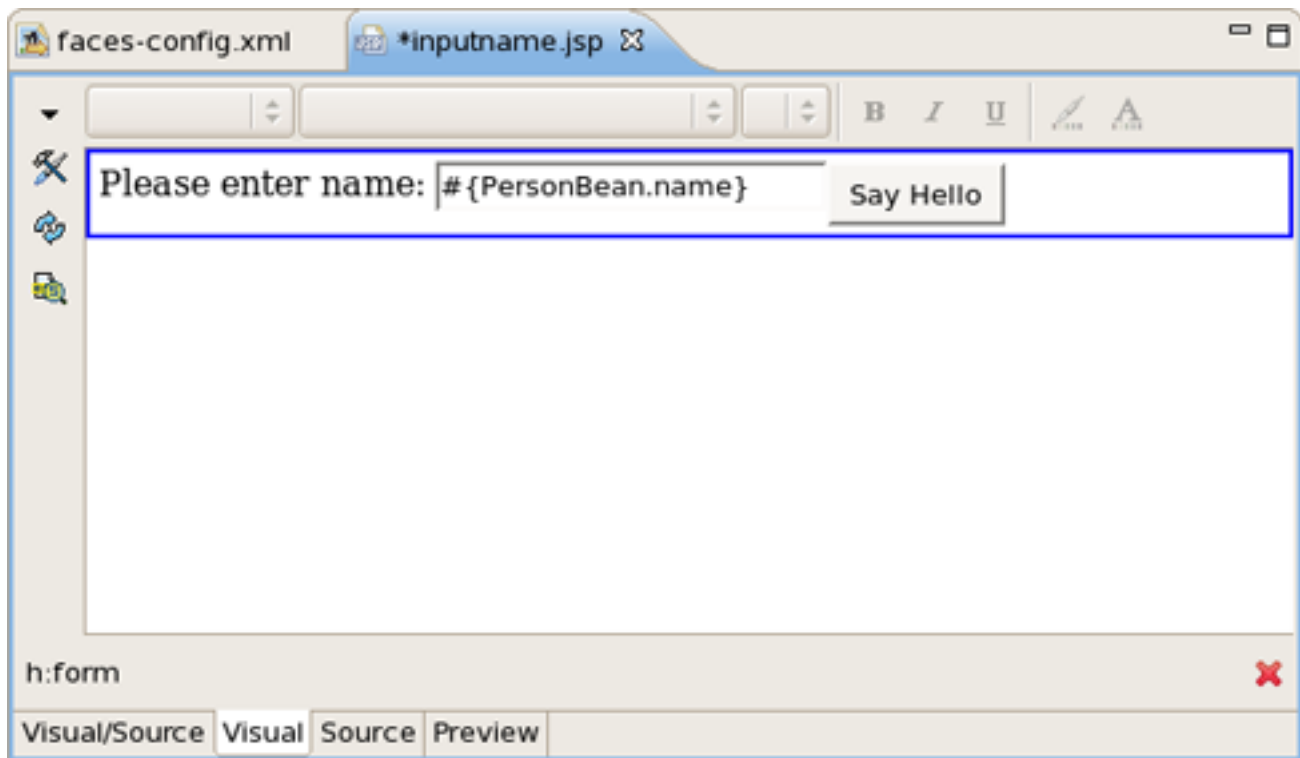


Figure 2.8. Visual Page Editor

- Save the file by selecting *File > Save* from the menu bar

2.6.2. greeting.jsp

- Click on the *faces-config.xml* tab to bring the diagram back
- Open the editor for the second file by double-clicking on the */pages/greeting.jsp* icon
- Select the *Visual* tab, so we can work with the editor completely in its WYSIWYG mode
- Type "Hello "(note space after Hello) into the box
- Select *outputText* within the JSF HTML palette folder and drag it into the innermost box in the editor after "Hello"
- In the attributes dialog, click in *value* field next to the value attribute and click on the ... (Browse) button
- Then, select the *Managed Beans > personBean > name* node, click on the *Ok* button, and then click on the *Finish* button
- Right after the output field, type an *exclamation point (!)*

The source coding should be something like this now:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
Hello <h:outputText value="#{personBean.name}" />!
</f:view>
</body>
</html>
```

- Save the file

2.7. Creating the Start Page

You also need to create a start page as an entry point into the application.

- In the Package Explorer view to the left, right-click *jsfHello > WebContent* and select *New > JSP File*
- For Name type in *index*, for Template select *JSPRedirect* and click *Finish*

A JSP editor will open up on the newly created file.

- In the Source part of the split screen, type */pages/inputname.jsf* in between the quotes for the page attribute

The source coding should look like this now:

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head></head>
<body>
<jsp:forward page="/pages/inputname.jsf" />
</body>
</html>
```

Note the *.jsf* extension for the file name. This is a mapping defined in the *web.xml* file for the project for invoking JavaServer Faces when you run the application.

- Select *File > Save* from the menu bar

2.8. Running the Application

Everything is now ready for running our application without having to leave JBoss Developer Studio by using the JBoss engine that comes with the JBoss Developer Studio plug-in. For controlling JBoss server within JBoss Developer Studio there is JBoss Server view:

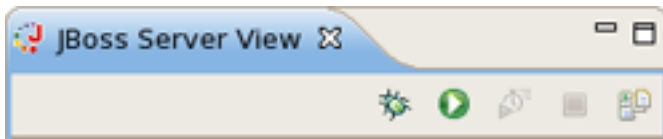


Figure 2.9. JBoss Server View

- Start up JBoss by clicking on the icon in JBoss Server view. (If JBoss is already running, stop it by clicking on the red icon and then start it again. Remember, the JSF run-time requires restarting the servlet engine when any changes have been made.) After the messages in the Console tabbed view stop scrolling, JBoss is available
- Click the Run icon or right click your project folder and select *Run As > Run on Server*:



Figure 2.10. Run Icon

This is the equivalent of launching the browser and typing `http://localhost:8080/jsfHello` into your browser. Our JSF application should now appear.

2.9. Other relevant resources on the topic

JSF on Sun: JavaServer Faces Technology [<http://java.sun.com/javaee/javaserverfaces/>]

Core JSF: Core JavaServer Faces [<http://www.horstmann.com/corejsf/>]

API: JSF API [<http://java.sun.com/javaee/javaserverfaces/1.1/docs/api/index.html>]

JSF Tags: JSF Core Tags [<http://www.horstmann.com/corejsf/jsf-tags.html>]

HTML Tags Reference: JSF HTML Tags Reference [<http://www.exadel.com/tutorial/jsf/jsftags-guide.html>]

JSF Central: JSF Central - Your JavaServer Faces Community [<http://www.jsfcentral.com/>]

FAQ: JSF FAQ [<http://wiki.java.net/bin/view/Projects/JavaServerFacesSpecFaq>]

Download: JavaServer Faces Technology - Download [<http://java.sun.com/javaee/javaserverfaces/download.html>]

Getting Started Guide for Creating a Struts Application

We are going to show you how to create a simple *Struts application* using the JBoss Developer Studio. The completed application will ask a user to enter a name and click a button. The resulting new page will display the familiar message, "Hello <name>!"

This document will show you how to create such an application from the beginning, along the way demonstrating some of the powerful features of JBoss Developer Studio. You will design the application, generate stub code for the application, fill in the stub coding, compile the application, and run the application all from inside JBoss Developer Studio.

We assume that you have already launched Eclipse with JBoss Developer Studio installed and also that the Web Development perspective is the current perspective. (If not, make it active by selecting *Window > Open Perspective > Other > Web Development* from the menu bar.)

3.1. Starting Up

We are first going to create a new project for the application.

- Go to the menu bar and select *File > New > Project....*
- Select *JBoss Tools Web > Struts > Struts Project* in the New Project dialog box
- Click *Next*
- Enter "StrutsHello" as the project name
- Leave everything else as is, and click *Next*
- Click *Next* again
- Make sure that *struts-bean.tld*, *struts-html.tld*, and *struts-logic.tld* are checked in the list of included tag libraries and then click *Finish*

A "StrutsHello" node should appear in the upper-left Package Explorer view.

- Click the plus sign next to *StrutsHello* to reveal the child nodes
- Click the plus sign next to *WebContent* under StrutsHello

- Click the plus sign next to *WEB-INF* under WebContent
- Then, double-click on the *struts-config.xml* node to display a diagram of the Struts application configuration file in the editing area

At this point, its empty except for the background grid lines.

3.2. Creating the Application Components

Now, we will design the application by creating the individual components as placeholders first. (We dont have to complete all of the details inside the components until afterwards.)

3.2.1. Creating JSP Page Placeholders

Next, let's create and place two JSP pages. We will not write any code for the files, but only create them as placeholders so that we can create links to them in the diagram. We will write the code a little bit later.

3.2.1.1. Creating the Page Placeholders

- Bring the Web Projects view to the front of the Package Explorer view by selecting the *Web Projects* tab next to that tab.
- Right-click the *StrutsHello > WEB-ROOT (WebContent)* folder in the Web Projects view and select *New > Folder...*
- Enter *pages* for a folder name and click *Finish*
- We will keep our presentation files in this folder
- Right-click the pages folder and select *New > File > JSP...*
- For Name type in *inputname* (the JSP extension will be automatically added to the file), for Template select *StrutsForm* and then click on the *Finish* button
- Right-click the pages folder again and select *New > File > JSP...*
- For Name type in *greeting*, for Template leave as Blank, and then click on the *Finish* button

Just leave these files as is for now.

3.2.1.2. Placing the Page Placeholders

Lets now place the two pages just created on the diagram.

- Click on the *struts-config.xml* tab in the Editing area to bring the diagram to the front
- Click on the *inputname.jsp* page in the Web Projects view, drag it onto the diagram, and drop it

- Click on the *greeting.jsp* page in the Web Projects view, drag it onto the diagram, and drop it to the right of the */pages/inputname.jsp* icon with some extra space

You should now have two JSP pages in the diagram.

3.2.2. Creating an Action Mappings

Using a context menu on the diagram, we are next going to create an Action mapping.

- Right-click between the two icons and select *Add > Action*
- Enter the following values:

Table 3.1.

path	/greeting
name	GetNameForm
scope	request
type	sample.GreetingAction
validate	<leave blank>

("GetNameForm" is the name for a form bean that we will create later.)

- Click *Finish*

The */greeting* action should appear in four places, in the diagram, under the action-mappings node, under the struts-config.xml node in Tree view, in Web Projects view and in the Outline view. Also, note the asterisk to the right of the name, struts-config.xml, in the Outline view showing that the file has been changed, but not saved to disk.

3.2.3. Creating a Link

Let's now create a link from the *inputname.jsp* page to the action.

- On the left-hand side of the diagram in the column of icons, click on this icon:



Figure 3.1. Create New Connection Icon

- In the connect-the-components mode you are in now, click on the */pages/inputname.jsp* icon in the diagram and then click on the */greeting* action

A link will be created from the page to the action.

3.2.4. Creating a Forward

Next, we are going to create a forward for the action.

- On the left-hand side of the diagram in the column of icons, click on this icon, again:



Figure 3.2. Create New Connection Icon

- Click on the */greeting* action icon in the diagram and then click on the *pages/greeting.jsp* icon
- That's it. A link will be drawn from the actions new greeting forward to the greeting.jsp JSP page. Note that the forwards name will be set based on the name of the target JSP file name. If you don't like it, you can easily change it
- Select the *Tree* tab at the bottom of the editor window (between Diagram and Source)
- Expand the *struts-config.xml/action-mappings//greeting* node and then select the greeting forward
- In the Properties Editor to the right, change the text to "sayHello" in the Name field
- Select the *Diagram* tab at the bottom of the editor window and see how the diagram is also updated to reflect the change

3.2.5. Creating a Global Forward

One last component that we need to create in the diagram is a global forward.

- Somewhere in the top-left corner of diagram, right-click and select *Add > Global Forward...*
- Enter *getName* in the Name field
- Select the *Change...* button for Path
- In the Edit Path window, switch to the *Pages* tab
- Expand the *StrutsHello > WEB-ROOT (WebContent) > pages* node and then select the *inputname.jsp* page
- Click *Ok*.
- Leave the rest of the fields blank and click *OK*

A forward object now appears on the diagram and also in the global-forwards folder in the Outline view.

- Tidy up the diagram, by clicking and dragging around each icon, so that the diagram looks something like this:

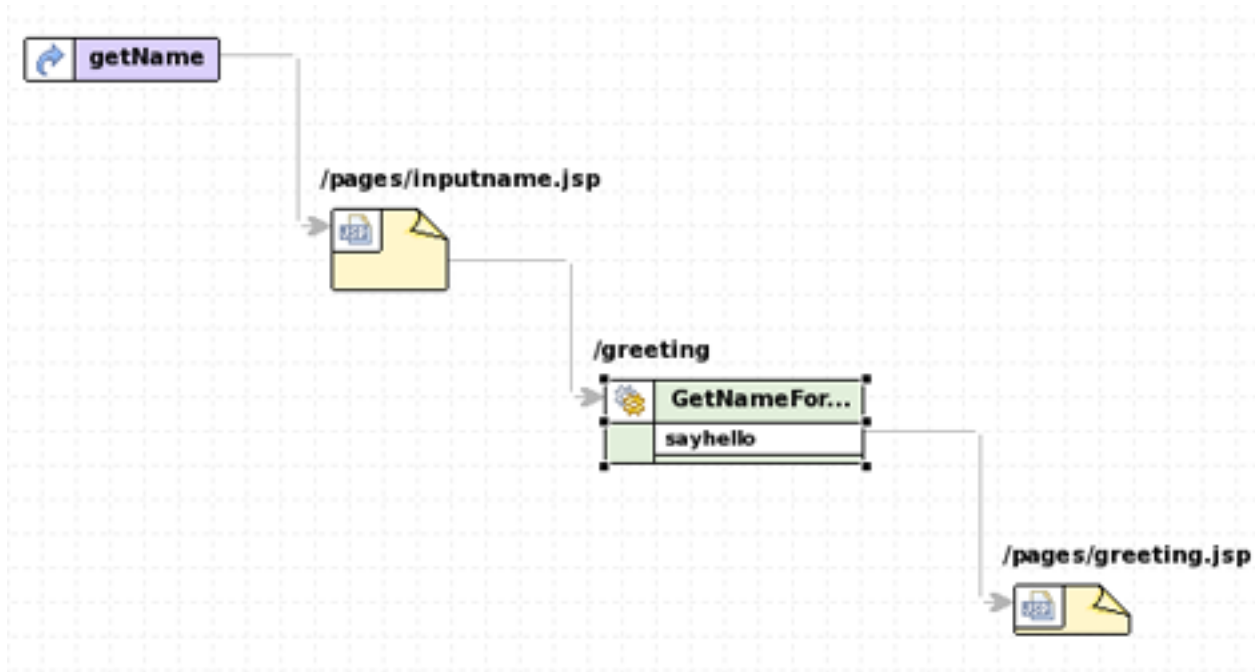


Figure 3.3. Diagram View

3.2.6. Creating a Form Bean

One last thing that we need to do is to create a form bean.

- Switch to the Tree viewer in the editor for the struts-config.xml file, by selecting the *Tree* tab at the bottom of the editor window
- Right-click *struts-config.xml* > *form-beans* and select Create Form Bean
- Enter *GetNameForm* in the name field and *sample.GetNameForm* for type
- Click *Finish*
- To save your changes to struts-config.xml, select *File* > *Save* from the menu bar

Note the disappearance of the asterisk next to the name, struts-config.xml.

3.3. Generating Stub Coding

We are done with designing the application through the diagram. Now we need to write code for the action component. We also need to write an action class for the */greeting* mapping along with a FormBean. To aid in the coding phase, JBoss Developer Studio can generate Java class stubs for all of the components shown in the diagram.

- Switch back to the diagram, by selecting the *Diagram* tab at the bottom of the editor window
- Right-click a blank space in the diagram and select *Generate Java Code*
- Leave everything as is in the dialog box and click *Generate*

You should see a screen that says:

Generated classes: 2

Actions: 1

Form beans: 1

- Click *Finish*

The Java files will be generated in a *JavaSource > sample* folder that you can see in the Package Explorer view under the "StrutsHello" node. One Action stub and one FormBean stub will have been generated.

3.4. Coding the Various Files

We will now code both the Java stub classes just generated, the JSP files left in as placeholders from previous steps, and a new start JSP page we will have to create.

3.4.1. Java Stub Classes

- To finish the two Java classes, switch to the *Package Explorer* view and expand the *JavaSource > sample* folder

3.4.1.1. GetNameForm.java

- Double-click *GetNameForm.java* for editing
- You are looking at a Java stub class that was generated by JBoss Developer Studio. Now we are going to edit the file
- Add the following attributes at the beginning of the class:

```
private String name = "";
private String greetName = "";
```

- Inside the reset method, delete the TO DO and throw lines and add:

```
this.name = "";
this.greetName = "";
```

- Inside the validate method, delete the TO DO and throw lines and add:

```
ActionErrors errors = new ActionErrors();
return errors;
```

- Right-click and select *Source > Generate Getters and Setters...* from the context menu
- In the dialog box, check the check boxes for name and greetName, select First method for Insertion point, and click on the *OK* button

The final GetNameForm.java file should look like this:

```
package sample;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;

public class GetNameForm extends org.apache.struts.action.ActionForm
{
    private String name = "";
    private String greetName = "";

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }

    public String getGreetName()
    {
        return greetName;
    }

    public void setGreetName(String greetName)
    {
        this.greetName = greetName;
    }

    public GetNameForm()
    {
    }

    public void reset(ActionMapping actionMapping, HttpServletRequest request)
    {
        this.name = "";
        this.greetName = "";
    }

    public ActionErrors validate(ActionMapping actionMapping, HttpServletRequest request)
    {
        ActionErrors errors = new ActionErrors();
        return errors;
    }
}
```

- Save the file

3.4.1.2. GreetingAction.java

- Open `GreetingAction.java` for editing
- Inside the `execute` method, delete the TO DO lines and add the following:

```
String name = ((GetNameForm)form).getName();
String greeting = "Hello, "+name+"!";
((GetNameForm)form).setName(greeting);
return mapping.findForward(FORWARD_sayHello);
```

The final version of `GreetingAction.java` should look like this:

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

public class GreetingAction extends org.apache.struts.action.Action
{
    // Global Forwards
    public static final String GLOBAL_FORWARD_getName = "getName";

    // Local Forwards
    public static final String FORWARD_sayHello = "sayHello";

    public GreetingAction()
    {
    }

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception
    {
        String name = ((GetNameForm)form).getName();
        String greeting = "Hello, "+name+"!";
        ((GetNameForm)form).setName(greeting);
        return mapping.findForward(FORWARD_sayHello);
    }
}
```

- Save the file
- Close the editors for the two Java files

The last thing left to do is to code the JSP files whose editors should still be open from having been created as placeholders.

3.4.2. JSP Pages

3.4.2.1. `inputname.jsp`

In this page, the user will enter any name and click the *submit* button. Then, the greeting action will be called through the form.

```
Input name:
```

- Click on the *inputname.jsp* tab in the Editing area to bring its editor forward
- In the Web Projects view, expand *StrutsHello > Configuration > default > struts-config.xml > action-mappings* and select */greeting*
- Drag it and drop it between the quotes for the "action" attribute to the **<html:form>** element in the Source pane of the editor
- Then type this text on a new line just below this line:
- Select the *Visual* pane of the editor
- Then, in the JBoss Tools Palette, expand the *Struts Form* library, select *text*, and drag it onto the box

Note:

By default there are only four groups on the JBoss Tools Palette. If you wish to make some group visible click the *Show/Hide* button on the top of palette and in the prompted dialog check the group (or groups) you want to be shown.

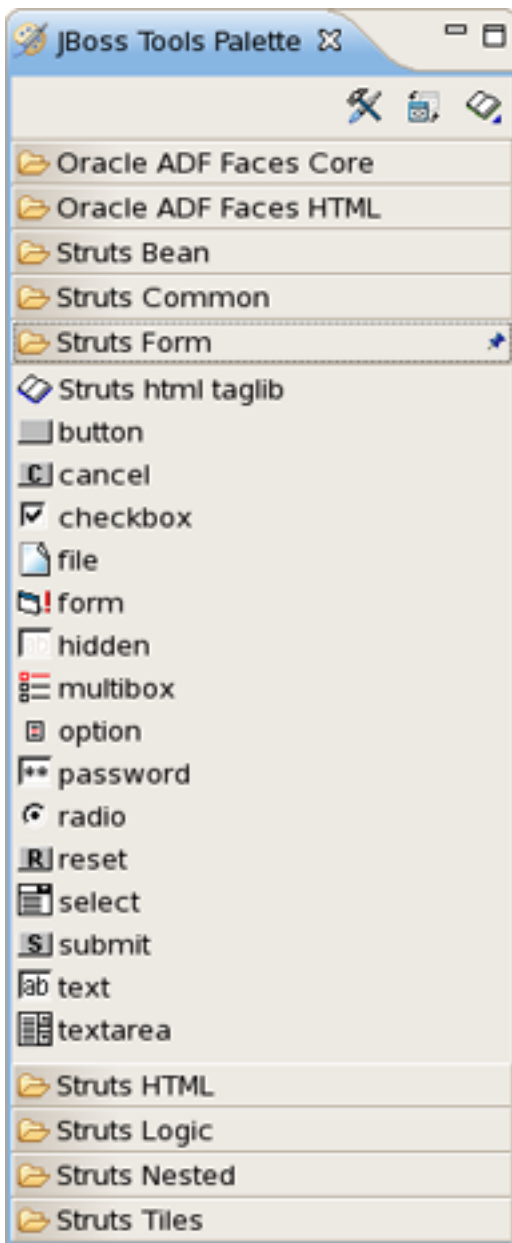


Figure 3.4. JBoss Tools Palette

- In the Insert Tag dialog box, type in name for property and select *Finish*
- In the StrutsForm library in the JBoss Tools Palette, select *submit*, and drag it to right after the text box in the Visual pane of the editor
- Right-click the *submit* button and select `<html:submit>` Attributes from the context menu
- In the Attributes dialog box, select the *value* field and type in "Say Hello!" for its value

After tidying the page source, the Editor window for the file should look something like this:

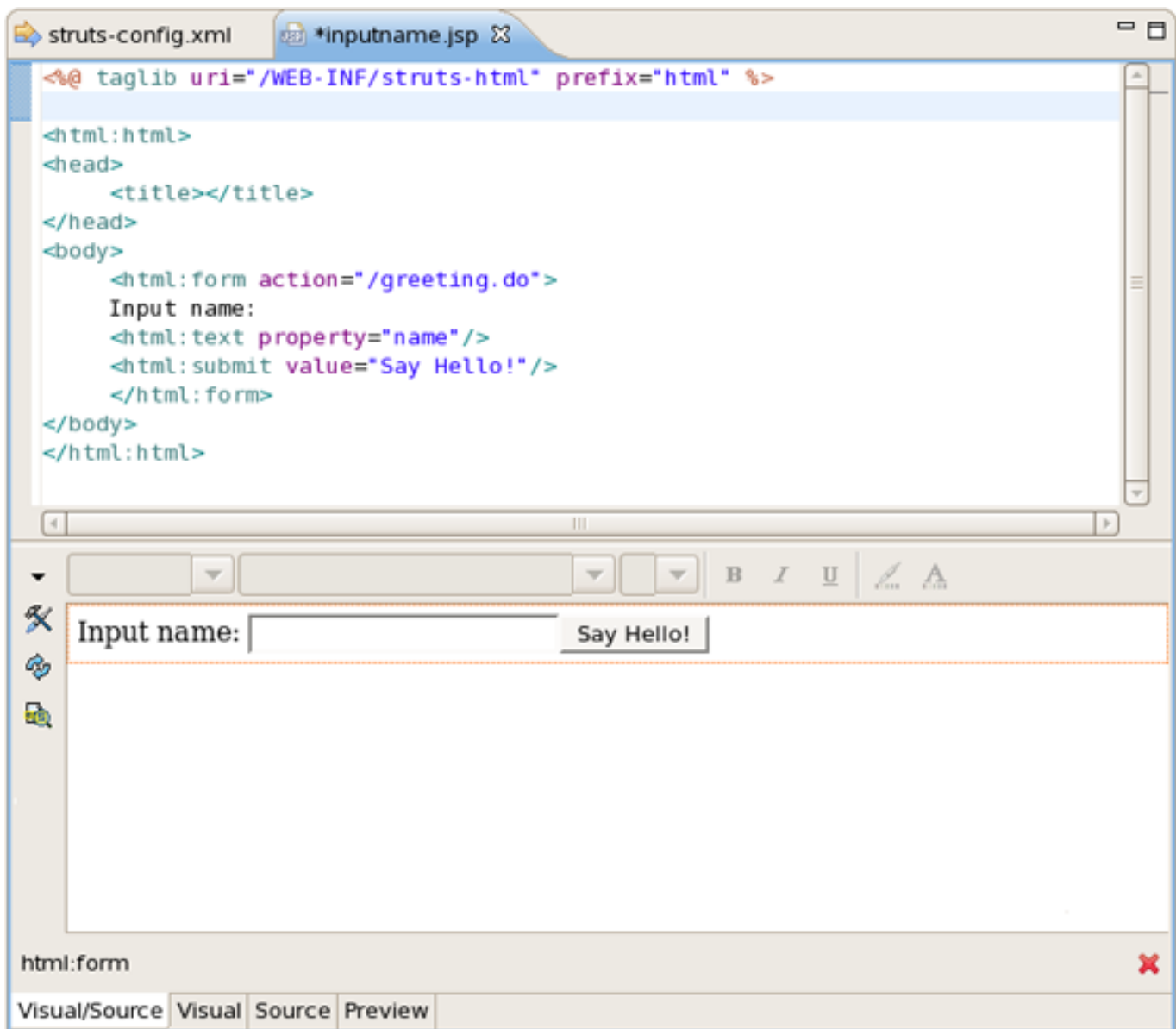


Figure 3.5. Editor Window

3.4.2.2. greeting.jsp

Next, we will fill in the result page.

- Click on the *greeting.jsp* tab in the Editing area to bring its editor forward
- Type in the following code:

```
<html>
<head>
  <title>Greeting</title>
</head>
<body>
  <p>
  </p>
```

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

To complete editing of this file, we will use macros from the JBoss Tools Palette. This palette is a view that should be available to the right of the editing area.

- Click on the *Struts Common* folder in the JBoss Tools Palette to open it
- Position the cursor at the beginning of the `greeting.jsp` file in the Source pane and then click on `bean taglib` in the JBoss Tools Palette

This will insert the following line at the top of the file:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
```

- Click on the *Struts Bean* folder in the JBoss Tools Palette to open it
- Position the cursor inside the `<p>` element
- Click on `write` in the JBoss Tools Palette
- Type in `"GetNameForm"` for the `"name"` attribute and add a `"property"` attribute with `"greetName"` as its value

The editor should now look like this:

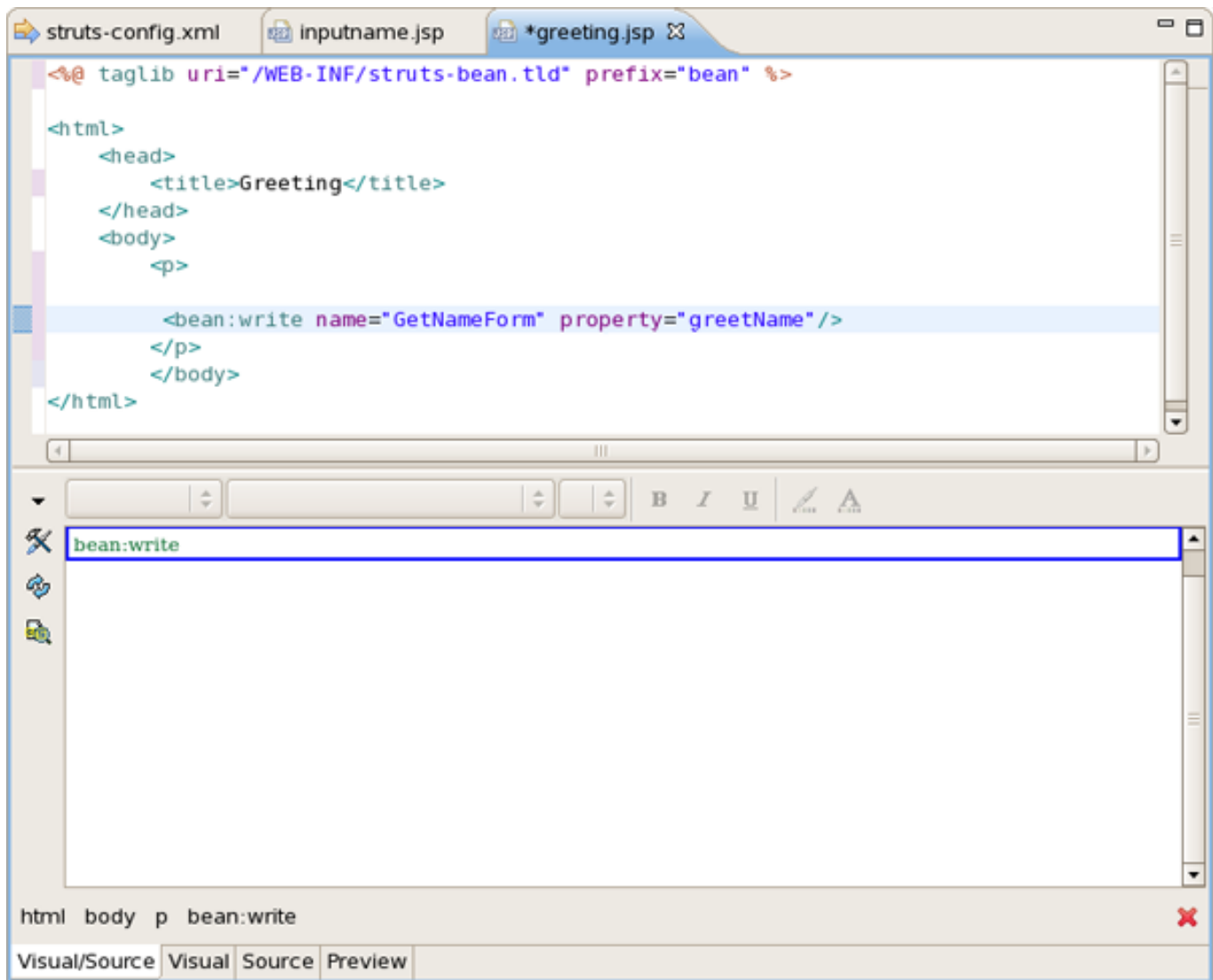


Figure 3.6. Editor Window

3.4.2.3. index.jsp

Finally, we will need to create and edit an index.jsp page. This page will use a Struts forward to simply redirect us to the getName global forward.

- In the Web Projects view, right-click on *StrutsHello* > *WEB-ROOT(WebContent)* node and select *New* > *File* > *JSP*..
- Type index for Name and click on the *Finish* button
- On the JBoss Tools Palette, select the *Struts Common* folder of macros by clicking on it in the palette
- Click on the logic taglib icon
- Press the *Enter* key in the editor to go to the next line

- Back on the palette, select the *Struts Logic* folder of macros
- Click on redirect
- Delete the ending tag, put a forward slash in front of the closing angle bracket, and type "forward=getName" in front of the slash

The finished code for the page is shown below:

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<logic:redirect forward="getName"/>
```

- To save all the edits to files, select *File>Save All* from the menu bar

3.5. Compiling the Classes

Because this is the Eclipse environment, no explicit compilation step is required. By default, Eclipse compiles as you go.

3.6. Running the Application

Everything is now ready for running our application without having to leave JBoss Developer Studio by using the JBoss Application Server engine that comes with the JBoss Developer Studio. For controlling JBoss AS within JBoss Developer Studio, there is JBoss Server view.

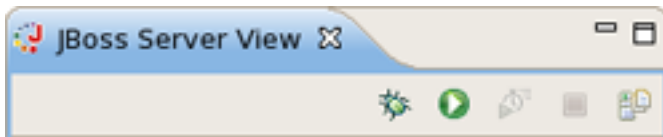


Figure 3.7. JBoss Server Panel

- Start up JBoss AS by clicking on the icon in JBoss Server view. (If JBoss AS is already running, stop it by clicking on the red icon and then start it again. Remember, the Struts run-time requires restarting the servlet engine when any changes have been made.)
- After the messages in the Console tabbed view stop scrolling, JBoss AS is available. At this point, right-click on the getName global forward in the struts-config.xml diagram view and select Run on Server.

The browser should appear with the application started.

3.7. Other relevant resources on the topic

Apache Struts: Struts Technology [<http://struts.apache.org/>]

Struts 2: Apache Struts 2 [<http://struts.apache.org/2.x/>]

Get Started: Struts Getting Started [<http://struts.apache.org/2.x/docs/home.html>]

Struts on IBM: Struts - An open-source MVC implementation [<http://www.ibm.com/developerworks/library/j-struts/>]

FAQ: Struts FAQ [<http://struts.apache.org/2.x/docs/faqs.html>]

Download: Release of Apache Struts [<http://struts.apache.org/download.cgi#struts206>]

4

Getting Started Struts Validation Examples

Validation of input is an important part of any Web application. All Apache Jakarta frameworks, including Struts, can use a common Jakarta Validation Framework for streamlining this aspect of Web application development. The Validation Framework allows you to define validation rules and then apply these rules on the client-side or the server-side.

JBoss Developer Studio makes using the Validation Framework in Struts even easier with the help of a specialized editor for the XML files that controls validation in a project. In this document, we'll show you how this all works by creating some simple client-side validation and server-side validation examples.

4.1. Starting Point

The example assumes that you have already created our sample "StrutsHello" application from the Getting Started Guide for Creating a Struts Application. You should have the JBoss Developer Studio perspective open on this StrutsHello project.

4.2. Defining the Validation Rule

In these steps you will set up the validation that can be used for either client-side or server side validation. You need to enable validation as a part of the project, define an error message, and tie it into an appropriate part of the application.

- Right-click on a "plug-ins" node under the *StrutsHello > Configuration > default > struts-config.xml* node in the Web Projects view and select *Create Special Plugin > Validators* from the context menu
- Further down in the Web Projects view, right-click on the *StrutsHello > ResourceBundles* node and select *New > Properties File...* from the context menu
- In the dialog box, click on the *Browse...* button next to the Folder field, expand the JavaSource folder in this next dialog box, select the sample subfolder, and click on the *OK* button
- Back in the first dialog box, type in "applResources" for the Name field and click on the *Finish* button
- Right-click on a newly created file and select *Add > Default Error Messages* from the context menu
- Drag up the sample.applResources icon until you can drop it on the resources folder under struts-config.xml
- Select *File > Save All* from the menu bar
- Select validation.xml under the *StrutsHello > Validation* node and double-click it to open it with the JBoss Tools

XML Editor

- Here you must create a Formset.
- In the validation.xml file editor click the button *Create Formset* on the panel *Formsets*
- In the dialog *Add Formset* fill the fields *Language* and *Country* or just leave them empty to create a default formset. Click *OK*

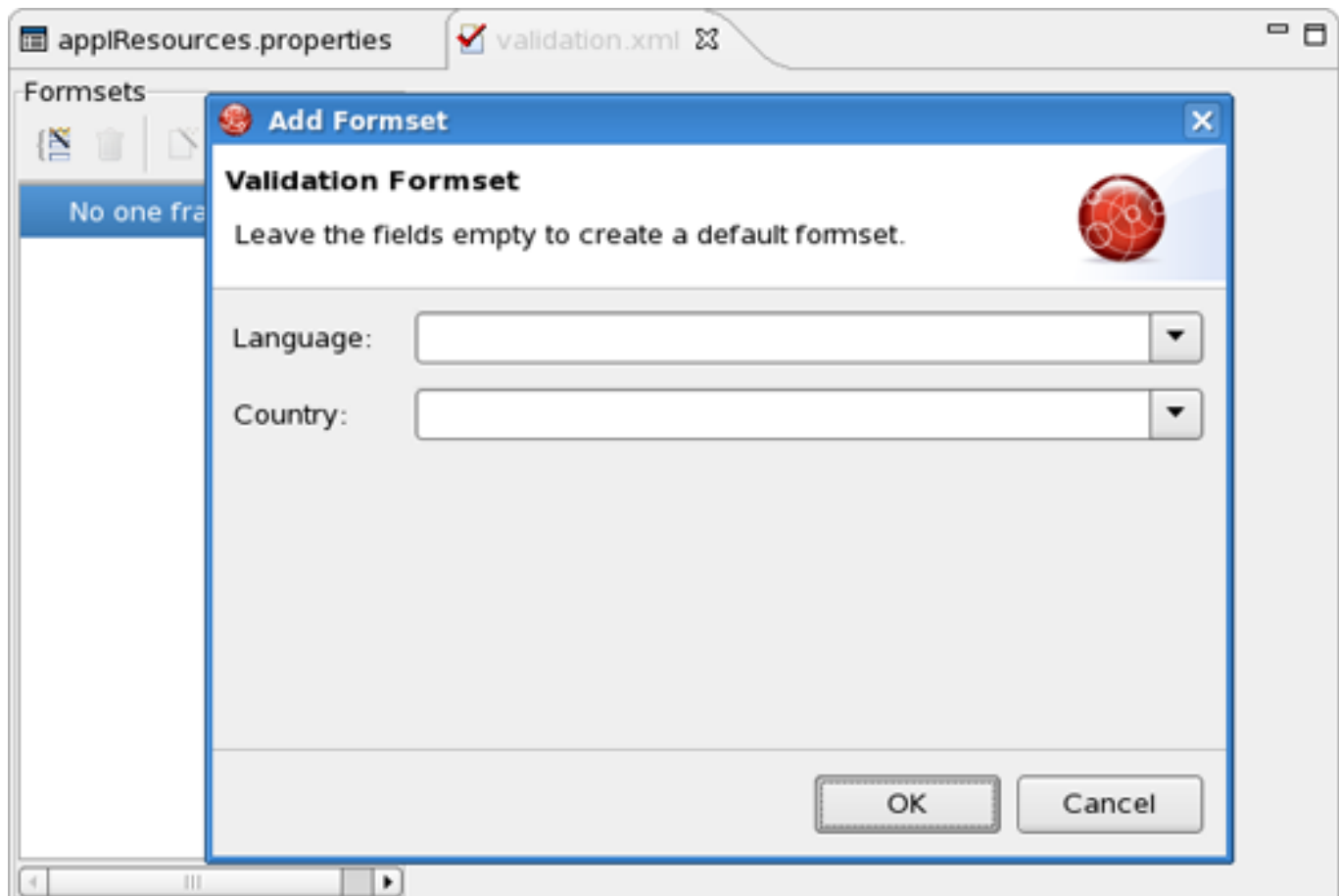


Figure 4.1. Create Formset

- Expand the "form-beans" node under the *StrutsHello > Configuration > default > struts-config.xml* node. Then, drag the form bean "GetNameForm" and drop it onto a formset in the XML Editor
- In the Validation Editor, expand the formset node, right-click GetNameForm, and select *Create Field...* from the context menu
- Enter a name for Property in the dialog box. A new property will be created:

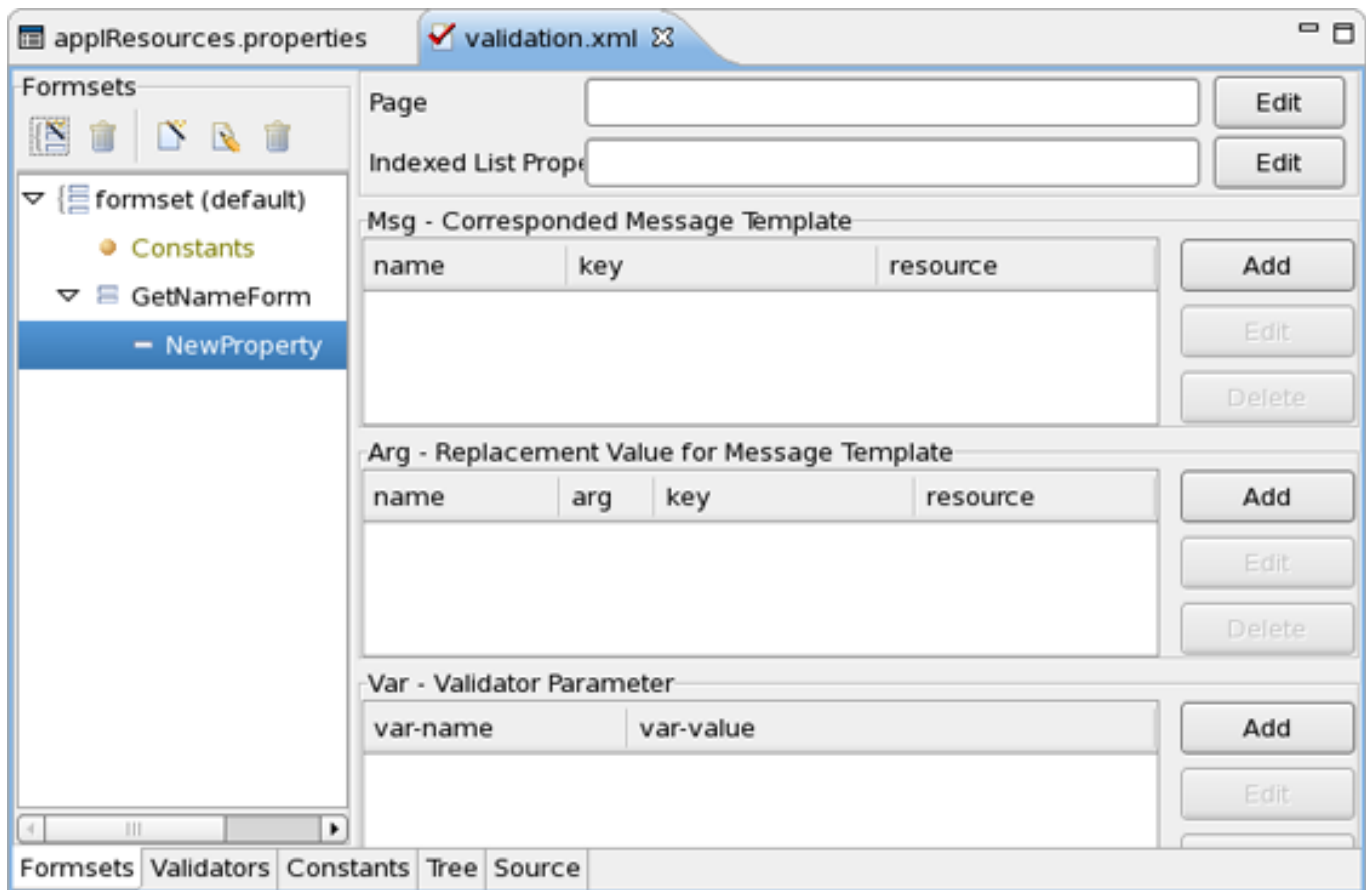


Figure 4.2. New Property Is Added

- In the Properties view for the name field to the right of the "tree" for the validation.xml file, click on the *Change...* button next to the *Depends* entry field
- In the displayed double list, select *required* from the left list and then click *Add*
- Click *Ok*
- Right-click name and select *Add Arg...* from the context menu
- In the Add Arg dialog box, click on the *Change...* button next to the *Key* field
- In the Key dialog box that appears now, click on the *Add* button
- Enter "name.required" in the Name field, and enter a person's name in the Value field
- Click *Finish*, then *Ok*, and then *Ok* again
- Select *File > Save All* from the menu bar

4.3. Client-Side Validation

Client-side validation uses a scripting language (like JavaScript) running in the client browser to actually do the validation. In a Struts application using the Validation Framework, however, you don't actually have to do any of the script coding. The Validation Framework handles this.

To see how this works in our application, you'll just need to make a couple of modifications to one of the JSP files.

- Double-click `inputname.jsp` under *StrutsHello > WEB-ROOT(WebContent) > pages* to open it for editing
- Find the tag near the top and hit Return to make a new line under it
- In the JBoss Tools Palette view to the right, open the Struts HTML folder and click on the javascript tag
- Back in the editor, just in front of the closing slash for this inserted tag, hit Ctrl+Space and select "formName" from the prompting menu
- Over in the Web Projects view, select `GetNameForm` under the *StrutsHello > Configuration > default > struts-config.xml > form-beans* node, drag it, and drop it between the quotes in the editor
- Modify the `<html:form>` tag by inserting this attribute:

```
onsubmit="return validateGetNameForm(this) "
```

The file should now look like this:

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html>
<head>
    <title>Input name</title>
    <html:javascript formName="GetNameForm" />
</head>
<body>
    <html:form action="/greeting.do" onsubmit="return
        <para>validateGetNameForm(this) "></para>
        <table border="0" cellspacing="0" cellpadding="0">
            <tr>
                <td><b>Input name:</b></td>
            </tr>
            <tr>
                <td>
                    <html:text property="name" />
                    <html:submit value=" Say Hello! " />
                </td>
            </tr>
        </table>
    </html:form>
</body>
</html:html>
```

- Select *File > Save* from the menu bar
- Start JBoss Application Server by clicking on its icon (a right-pointing arrow) in the toolbar
- Click the Run icon or right click your project folder and select *Run As > Run on Server*



Figure 4.3. Run Icon

- In the browser window, click on the "Say Hello!" button without having entered any name in the form
- A JavaScript error message should be displayed in an alert box.

4.4. Server Side Validation

Server side validation does the validation inside the application on the server. In a Struts application using the Validation Framework, you still don't have to do any of the actual validation coding. The Validation Framework handles this. You will though have to make a few changes to the JSP file you modified for client-side validation along with a change to an action and a few changes to the form bean class.

4.5. Editing the JSP File

- Reopen `inputname.jsp` for editing
- Delete the `onsubmit` attribute in the `<html:form>` element that you put in for client-side validation
- Add an `<html:errors>` tag after the `<html:form>` tag

The JSP file should now look like this:

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html>
<head>
  <title>Input name</title>
  <html:javascript formName="GetNameForm" />
</head>
<body>
  <html:form action="/greeting.do">
    <table border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td><b>Input name:</b></td>
      </tr>
      <tr>
        <td>
          <html:text property="name" />
          <html:submit value=" Say Hello! " />
        </td>
      </tr>
    </table>
  </html:form>
  <html:errors />
</body>
</html:html>
```

4.6. Editing the Action

- In the Web Projects view, expand the node under the *StrutsHello* > *Configuration* > *default* > *struts-config.xml* > *action-mappings* node, right-click the */greeting* action, and then select *Properties...* from the context menu
- In the Edit Properties window, insert the cursor into the value column for the input property and click on the ... button
- In the dialog box, make sure the Pages tab is selected, select *StrutsHello* > *WEB-ROOT(WebContent)* > *pages* > *inputname.jsp*, click the *Ok* button, and then click on the *Close* button

4.7. Editing the Form Bean

- Right-click the */greeting* action again and select Open Form-bean Source to open the *GetNameForm.java* file for editing
- Change the class that it extends from: *org.apache.struts.action.ActionForm* to *org.apache.struts.validator.ValidatorForm*
- Comment out a validate method

The file should now look like this:

```
package sample;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
public class GetNameForm extends
org.apache.struts.validator.ValidatorForm
{
    private String name = "";

    /**
     * @return Returns the name.
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name The name to set.
     */
    public void setName(String name)
    {
        this.name = name;
    }

    public GetNameForm ()
    {
    }

    public void reset(ActionMapping actionMapping,
```

```
        HttpServletRequest request)
    {
        this.name = "";
    }

    // public ActionErrors validate(ActionMapping actionMapping,
    //                               HttpServletRequest request)
    {
        // ActionErrors errors = new ActionErrors();
        // return errors;
        // }
    }
```

- Select *File > Save All* from the menu bar
- Reload the application into JBoss AS by clicking on the "Change Time Stamp" icon (a finger pointing with a little star) in the toolbar
- Run the application
- In the browser window, click on the "Say Hello!" button without having entered any name in the form

The error message should appear in a refreshed version of the form.

4.8. Other Resources

You can also read Struts [<http://www.redhat.com/developers/jbds/JSFTools/Struts.html>] chapter in our "JBoss JSF Tools" guide for more information on this topic.