

Build Apache Geronimo applications using JavaServer Faces, Part 2: Using Tomahawk with JavaServer Faces

Skill Level: Intermediate

[Chris Herborth](#)

Freelance

Freelance Writer

26 Sep 2006

This tutorial, Part 2 of a five-part series, introduces you to Apache Tomahawk. In the [first installment](#) of this series, you learned about Java™Server Faces™ (JSF), a new Java-based framework that makes it easier to build graphical user interfaces (GUIs) for Java Platform, Enterprise Edition (Java EE) applications, and you built and successfully deployed a simple JSF application on Apache Geronimo. Now you'll explore Apache Tomahawk -- which provides several custom, 100% JSF-compatible components -- and learn how to integrate it with your application to improve the interface.

Section 1. Before you start

This tutorial shows Java programmers how to build highly interactive Java EE applications for deployment on Apache Geronimo using the JSF components. The tutorial assumes you'll be using the Eclipse IDE as your development platform.

About this tutorial

This tutorial introduces you to Apache Tomahawk, a set of complementary components that will improve the interface of your JSF application. You'll continue developing the front end for the sign-up pages of the developer forum by adding some input validators and additional input widgets.

About this series

This tutorial is Part 2 of a five-part series about building Apache Geronimo applications using JSF. Here's a rundown of the entire series:

- **Part 1: Use Eclipse and Apache MyFaces Core to build a basic application** introduced you to using Apache's MyFaces implementation of the JSF standard with Geronimo, a free application server (also from Apache). This tutorial showed you how to use the Eclipse IDE's Web Tool Platform (WTP) to build JSF applications.
- **Part 2: Using Tomahawk with JavaServer Faces** shows you how to integrate Apache Tomahawk components with your Geronimo application. Tomahawk provides several custom components that are 100% compatible with JSF.
- **Part 3: Using Ajax4jsf with JavaServer Faces** demonstrates how to use Sun's free open source framework, Ajax4jsf, to add Asynchronous JavaScript + XML (Ajax) capabilities to your Geronimo application.
- **Part 4: Extend JSF with Apache Trinidad components** teaches you how to integrate components from Apache Trinidad, the open source version of ADF Faces, with your Geronimo application to enhance your JSF application's interface.
- **Part 5: Integrating your JSF Application with Spring** shows you how to integrate your JSF applications with the Spring Framework, a popular framework that makes it easier for Geronimo developers to build Java EE applications.

System requirements

You need the following tools to follow along with this tutorial:

- [Geronimo](#), Apache's Java EE server project. Geronimo comes in Tomcat and Jetty flavors, depending on your needs. We used the Jetty flavor (version 1.1) because it's smaller.
- [MyFaces](#), Apache's JSF implementation. Download the core version (without Tomcat) from Apache. We used version 1.1.3 with this tutorial.
- [Tomahawk](#), which provides additional components and input validators for use with MyFaces, while still maintaining 100% JSF compatibility.
- [Eclipse](#), the extensible open source IDE that supports a wide range of languages and platforms.

- [Eclipse Web Tools Platform \(WTP\)](#), which adds support for XML and JavaScript editing, as well as preliminary JSF support, to Eclipse.
 - [Java 1.4 or newer](#) installed on your system. Eclipse binaries come with their own Java run time, but Geronimo and MyFaces don't (that would seriously bloat up the download archives). We use Java 1.5 on Mac OS X 10.4 in this tutorial, but the platform shouldn't matter. Get Java technology from [Sun Microsystems](#) or [IBM®](#).
-

Section 2. Overview

When creating Web applications, Java Servlets and JavaServer Pages™ (JSP™) technology offer a powerful environment, but present no standard way of creating the UI. You're on your own for managing the state of any forms in your JSP pages, and you must dispatch incoming HTTP requests to their proper event handlers. If your site has a complex GUI, the complex infrastructure that grows around your application will eventually take on a life of its own, and site-specific behavior and other details will creep in, making it difficult to reuse any of the code you build. JSF offers an off-the-shelf tool to simplify high-level tasks (such as arranging and reusing UI components) and connected component state and input handling with the objects that define your application's behavior.

Apache Geronimo

Apache Geronimo is an open source (licensed under the Apache Software Foundation's license) Java EE server designed for maximum compatibility and performance. The current version (1.1 as of this writing) passes Sun's Technology Compatibility Kit (TCK) for Java 2 Platform, Enterprise Edition (J2EE™) 1.4 servers, meaning that it's a fully compatible J2EE server per Sun's specifications.

Packaged with the Jetty or Tomcat Web server, Geronimo is easy to get up and running, and has an extremely useful management interface application already deployed. You can upload and activate Web applications from the management console without needing to restart or reconfigure the server in any way.

Apache MyFaces

Apache MyFaces is the first free open source implementation of the JSF Web application framework. JSF is similar to the popular Struts framework and

implements a Model-View-Controller (MVC) pattern, but has features that go beyond what Struts provides. JSF is defined in Java Specification Request #127 (JSR 127), a Java Community Process (JCP) spec that's been ratified by experts from across the Web application industry.

Apache Tomahawk

Apache Tomahawk provides a set of JSF components that go beyond the ones included in the specification, while being 100% compatible with any conforming JSF 1.1 implementation (such as MyFaces).

Eclipse

Eclipse is an open source integrated development environment (IDE) built around an extensible, plug-in-oriented architecture. This makes it possible for one IDE to support almost any language, task, platform, or data file you might need for doing almost any kind of work. In this case, you'll take advantage of the outstanding Java development support and the WTP project's plug-ins. The WTP provides editor support for XML and has experimental MyFaces support.

Let's take a quick look at the sample application.

The example application

In [Part 1](#), you used Geronimo to deploy a simple Web application written using MyFaces, intended to give you a good start for creating your own Web applications. The application handles the sign-up process for a fictional developer-oriented discussion forum.

In the next section, you'll review everything you did in the first tutorial, then you'll look at Tomahawk's components and extend the sign-up application to take advantage of a couple of them.

Section 3. Review

In [Part 1](#), you were introduced to Apache Geronimo and the MyFaces implementation of the JSF standard. You learned how to use the Eclipse IDE to create a new JSF project, export it as a .war file, and deploy it on the Geronimo

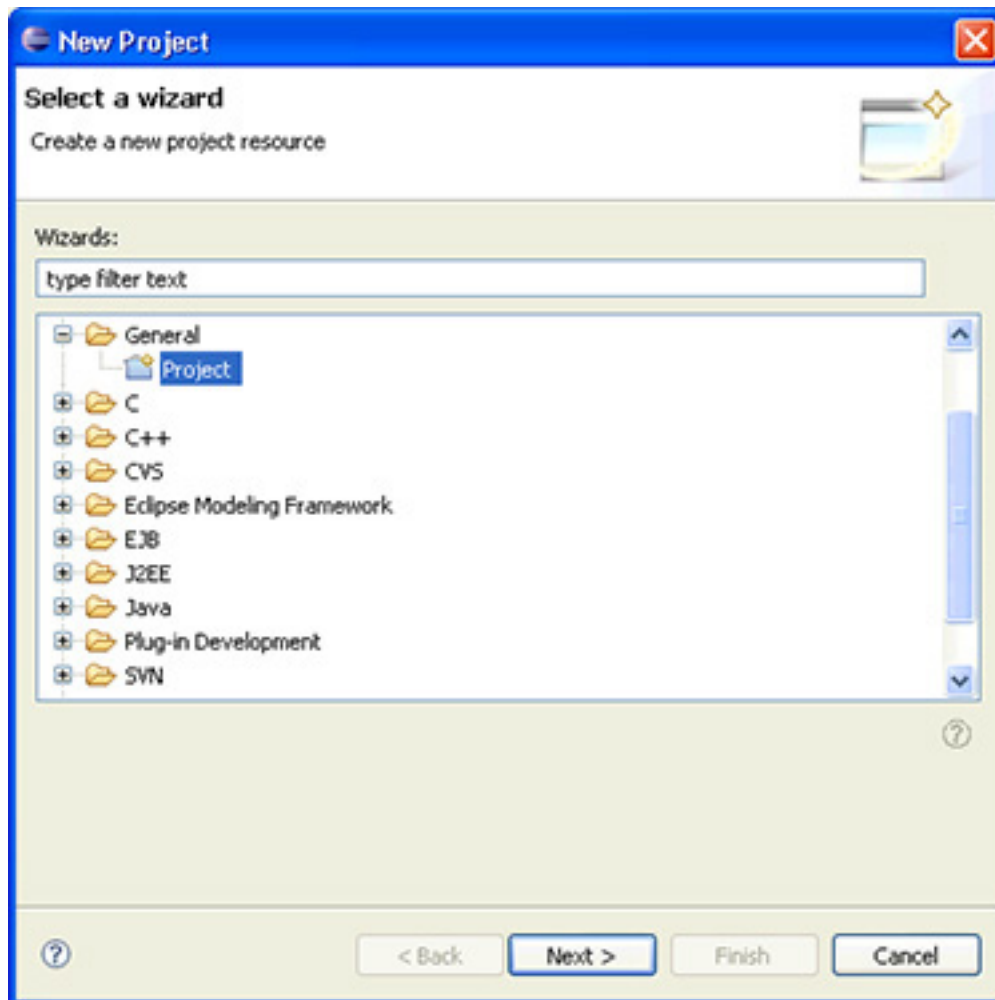
server. You also saw how to use the Faces Configuration editor in Eclipse to connect JSP processing results with appropriate response files.

Import the devSignup project

You should download the devSignup sample project from [Part 1](#), because you'll be adding to it later in this tutorial.

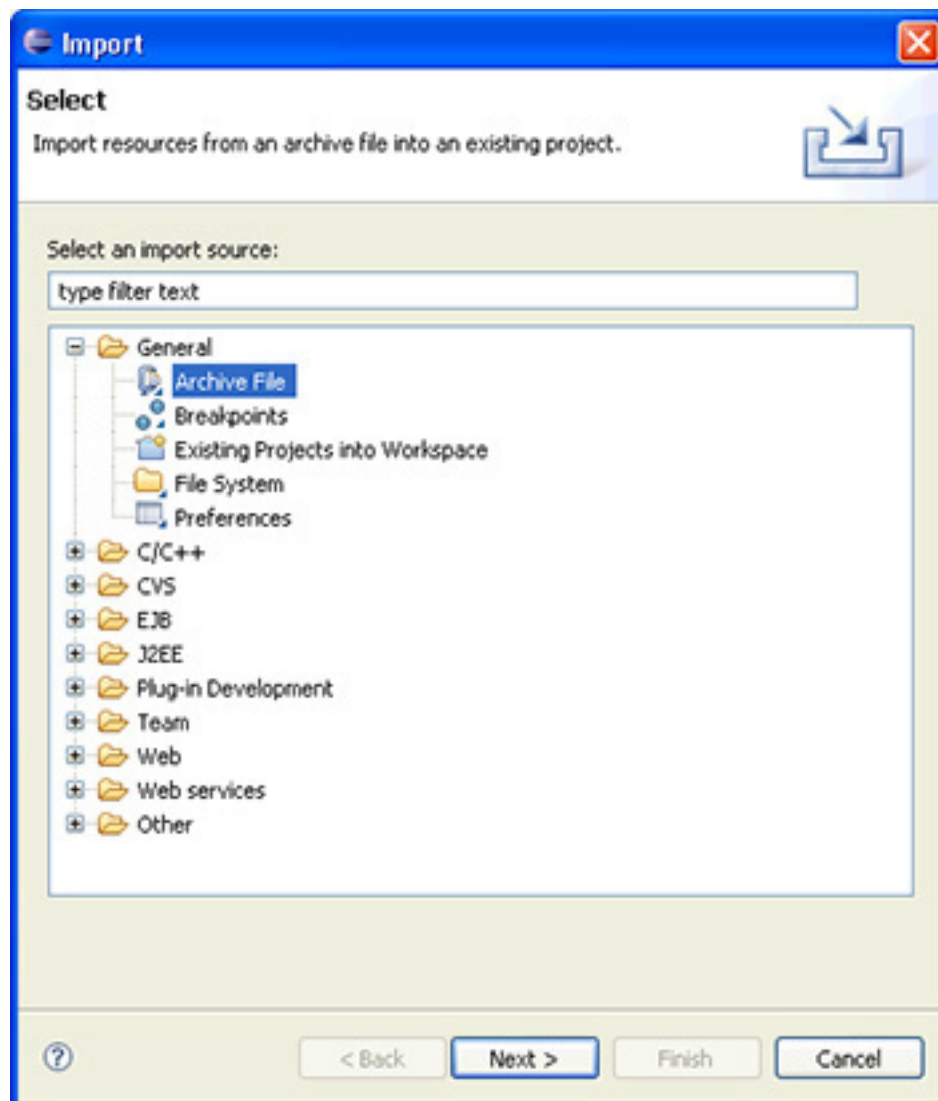
1. Launch Eclipse, and, if you haven't already, install the Web Tools Platform and its dependencies using the Update Manager. Be sure to click the **Select Required** button in the Update Manager after selecting Web Tools Platform from the list of plug-ins; this will ensure that you have the required parts of at least two other projects (the Eclipse Modeling Framework and Graphical Editing Framework).
2. Choose **File > New Project** from the menu to display the New Project dialog box (see [Figure 1](#)).

Figure 1. Creating a new project Eclipse



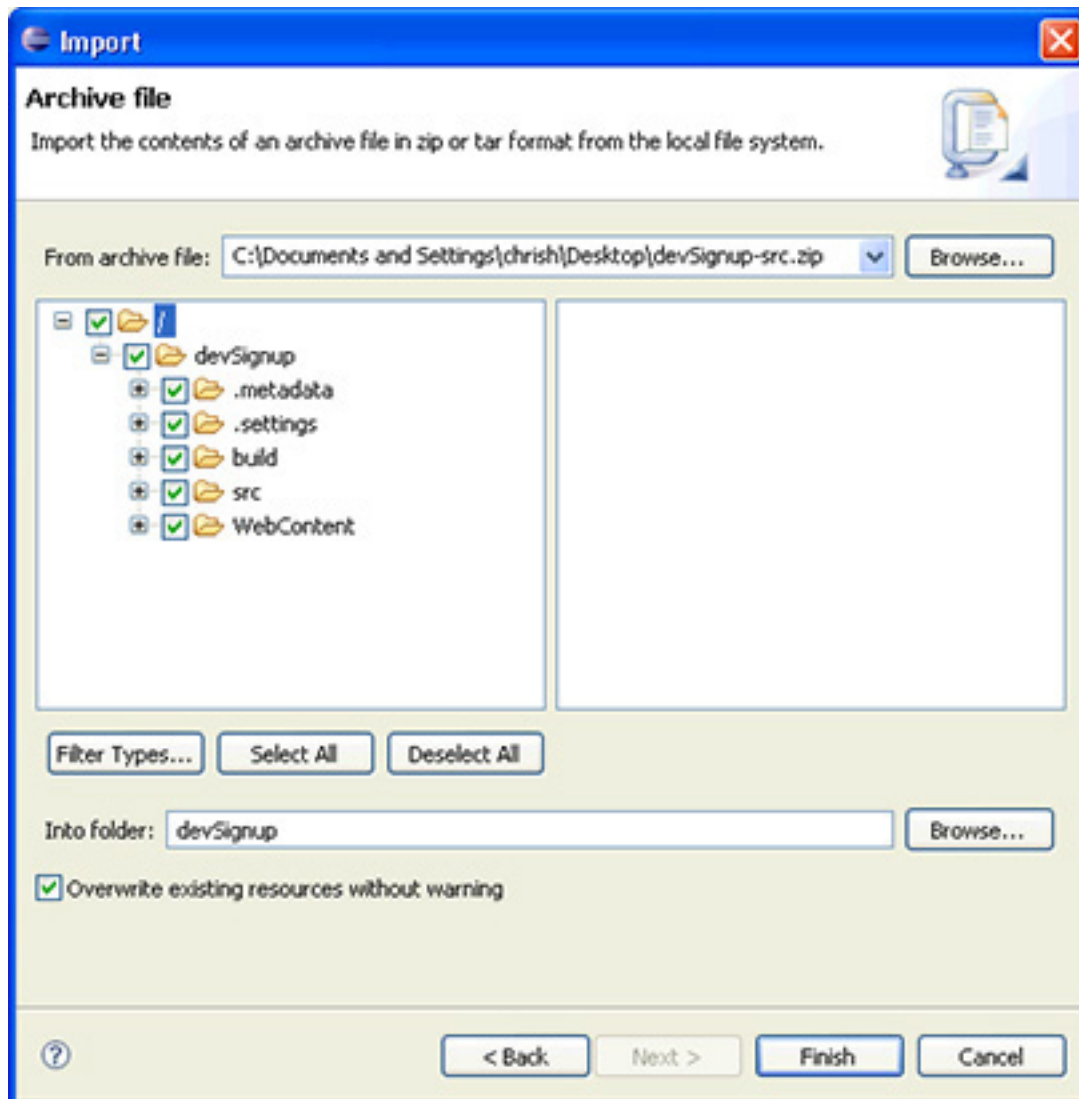
3. Expand the **General** category, and choose **Project**.
4. Click **Next**.
5. Enter `devSignup` as the project name, then click **Finish** to create the empty project.
6. Right-click your new **devSignup** project in Eclipse's Navigator view, then choose **Import** from the context menu to display the Import dialog (see [Figure 2](#)).

Figure 2. Importing the files for the devSignup project



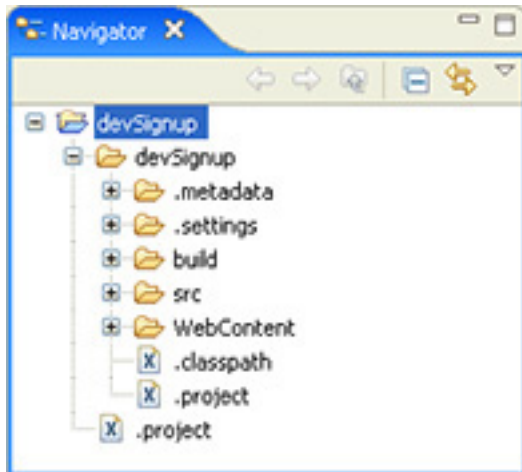
6. Expand the **General** category, and choose **Archive File** (because the project is provided as the archive, devSignup-src.zip).
7. Click **Next**.
8. Click the **Browse** button, and select the devSignup project's archive (see [Figure 3](#)).

Figure 3. The devSignup project comes in a ZIP archive



9. Click **Finish** to import the devSignup Web application's source files. Looking in the Navigator view, you'll notice that all of the files are one level too deep, below an extra devSignup folder (see [Figure 4](#)).

Figure 4. Why are my files in a subfolder?



Unfortunately, this is common with Eclipse.

10. Select everything below the extra folder (the `.metadata`, `.settings`, `build`, `src`, and `WebContent` directories, along with `.classpath` and `.project`) and drag them up to the top-level `devSignup` folder.
11. Delete the spare (and empty) **devSignup** folder, and you're all set.

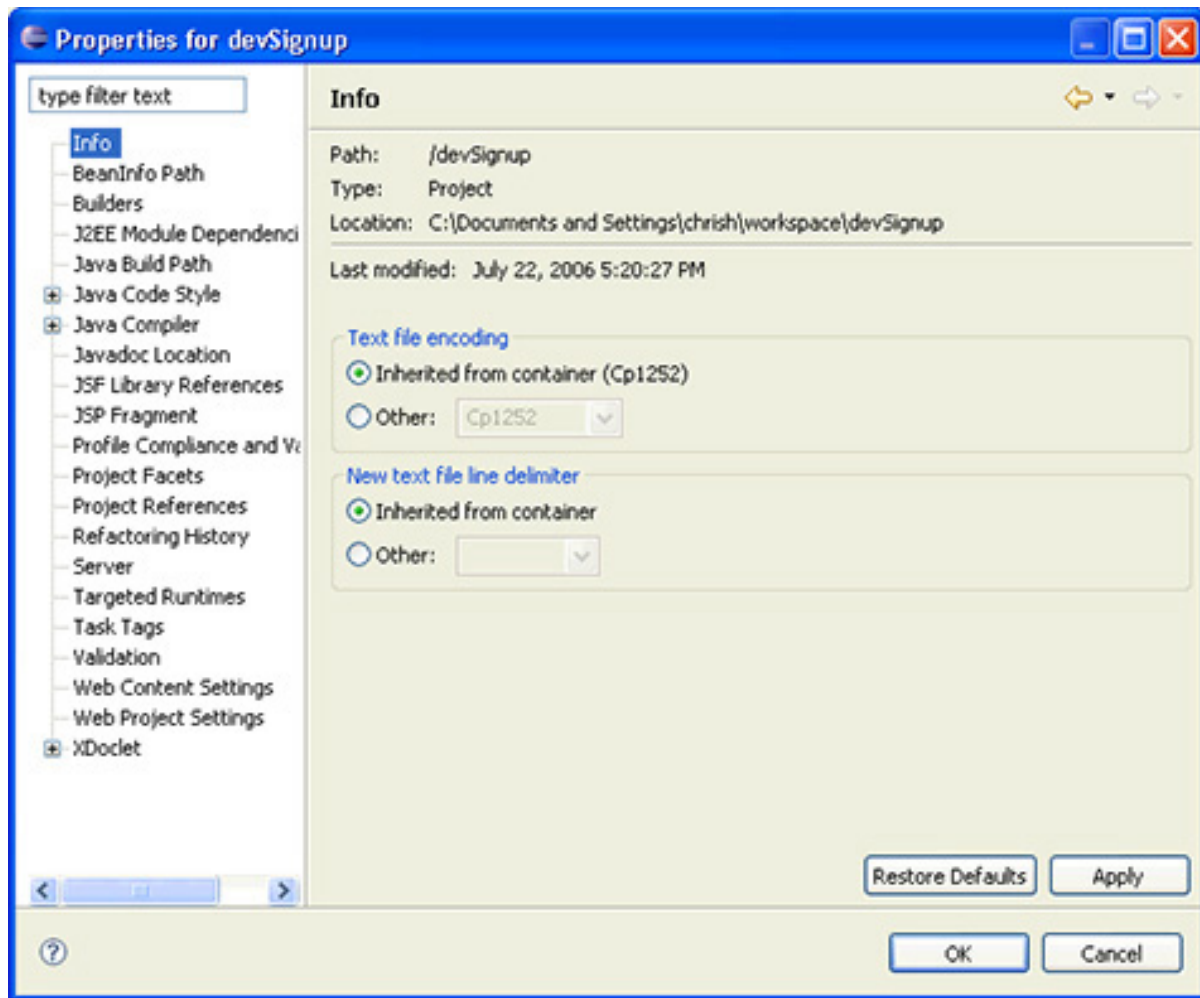
When you overwrite the top-level `.project` file, Eclipse notices that this is actually a Web application project and validates all of your files. If it asks you for permission to cache files from Sun's developer Web site, let it; Eclipse is going to use those files to validate the Web project while you're working. (Well, almost. Unless your system happens to be set up exactly like the iBook I do most of my work on -- hey, don't laugh, it's small and light -- Eclipse isn't going to be able to find the MyFaces `.jar` files it's going to need to deploy this project.)

Fix the devSignup project

Certain parts of the `devSignup` project contain system-specific information, such as the paths to the installed Java libraries and the MyFaces JARs -- again, a common problem when sharing projects with Eclipse. Let's go through the project's settings and fix up the bits that need fixing.

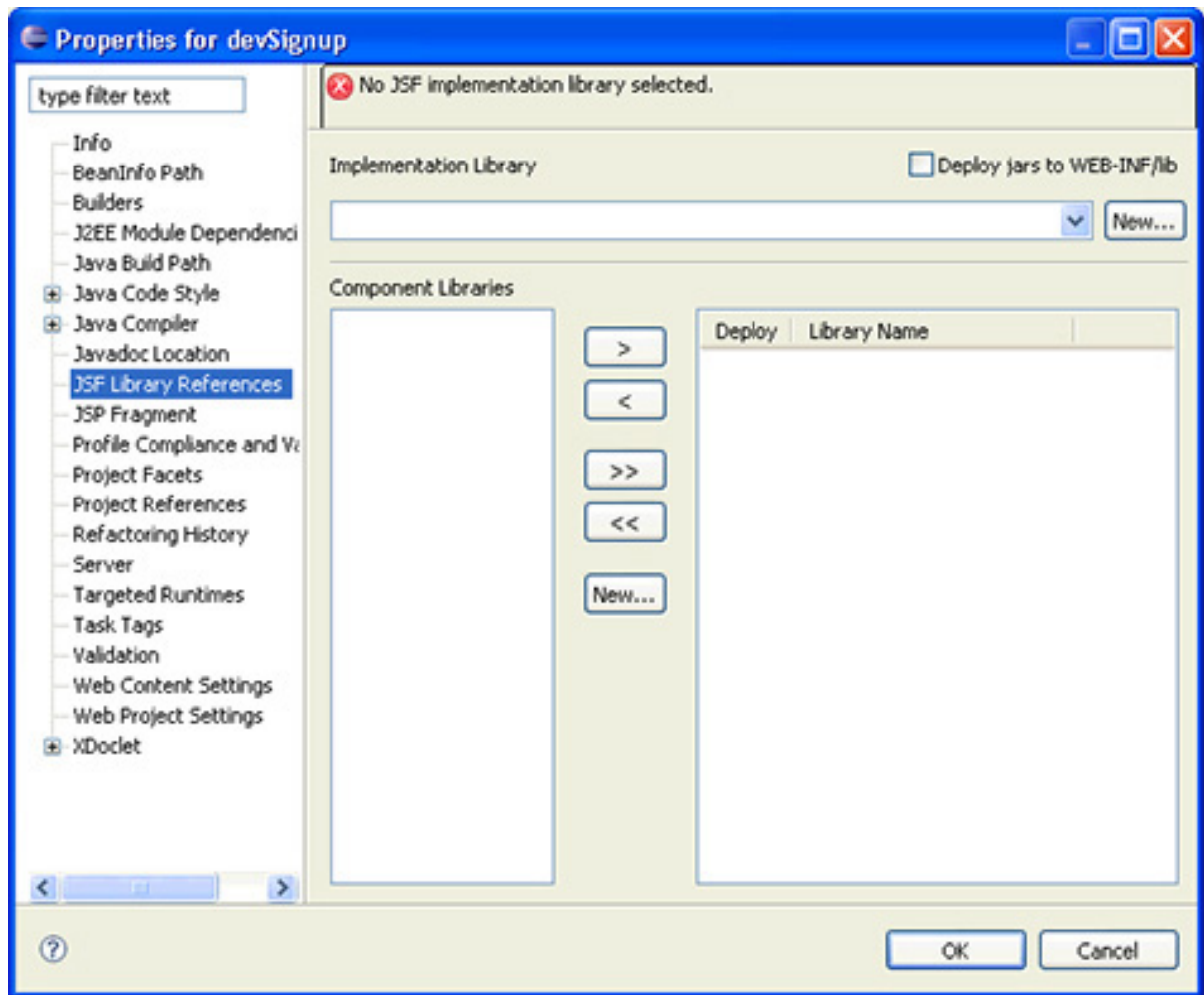
1. Right-click the **devSignup** project, and choose **Properties** from the context menu. Eclipse displays the Properties dialog, as shown in [Figure 5](#).

Figure 5. Eclipse's project properties for the devSignup project



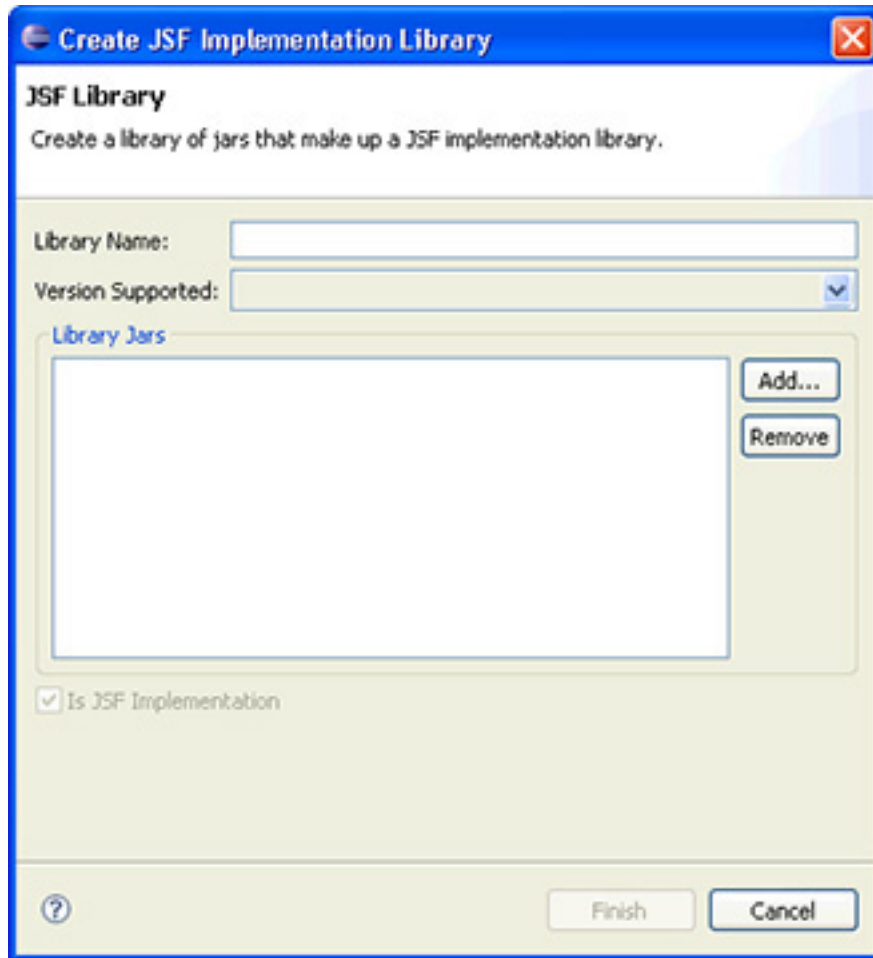
2. Click the **JSF Library References** entry in the list; you're going to need to tell the project where your MyFaces files are. Eclipse displays the JSF Library References properties for this project (see [Figure 6](#)).

Figure 6. Setting up a JSF library reference



3. At the top of the panel, check **Deploy jars to WEB-INF/lib**, because you definitely want the JSF libraries included when you create your application.
4. Click the **New** button right below this check box to display the Create JSF Implementation Library dialog box, as shown in [Figure 7](#).

Figure 7. Creating a JSF Implementation library



5. Give the library a descriptive name; I'll be using `MyFaces (Core)`.
6. Select `v1_1` from the Version Supported drop-down list (because Apache MyFaces supports version 1.1 of the JSF specification), and then click the **Add** button to select the JSF .jar files to include in this library definition.
7. Include the following files from the MyFaces Core distribution:
 - commons-beanutils-1.7.0.jar
 - commons-codec-1.3.jar
 - commons-collections-3.1.jar
 - commons-digester-1.6.jar
 - commons-el-1.0.jar
 - commons-lang-2.1.jar
 - commons-logging-1.0.4.jar

- jstl-1.1.0.jar
- myfaces-api-1.1.3.jar
- myfaces-impl-1.1.3.jar

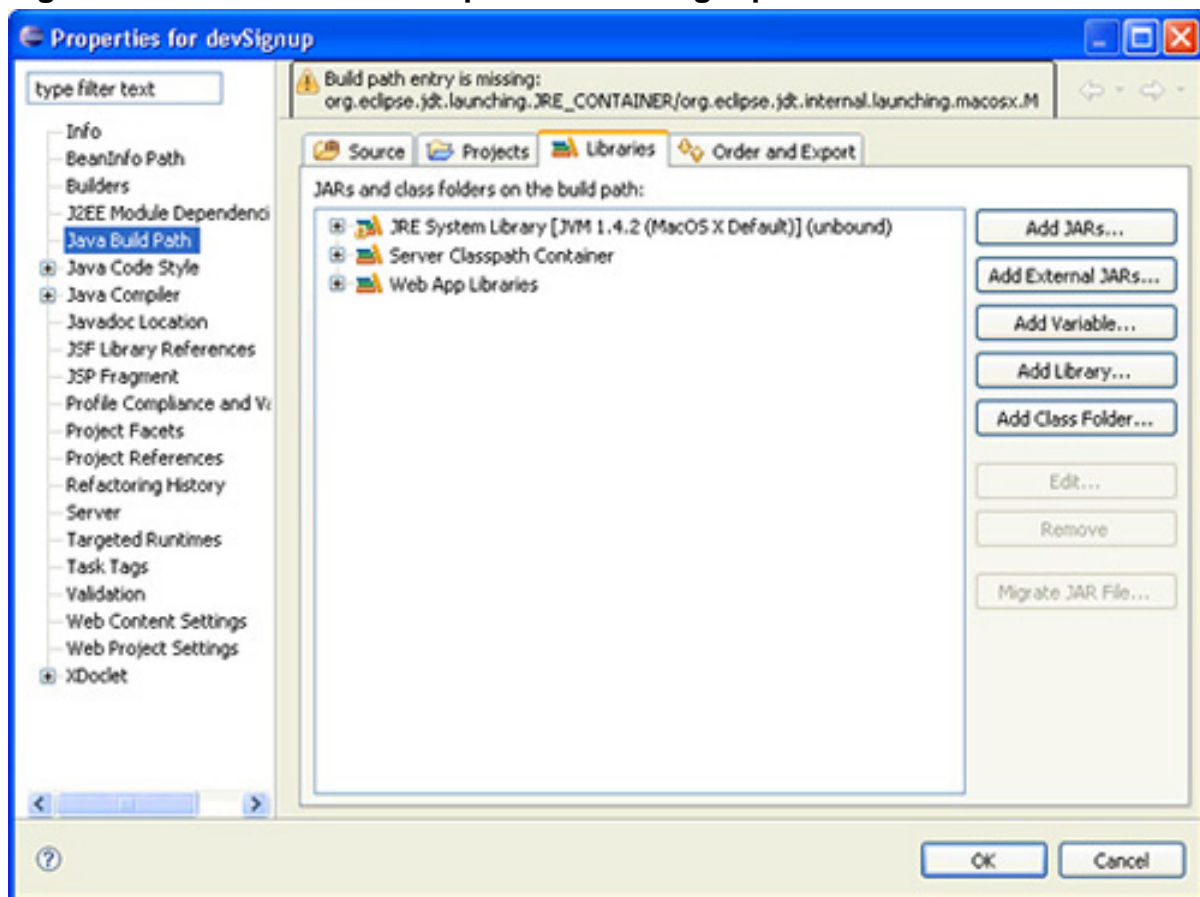
This is the current minimal set of JSF libraries in the MyFaces implementation (some documents leave jstl.jar out of the list, but it *is* required).

One thing to note here is that the JSF Implementation library is going to contain full path references to your .jar files; it doesn't copy them into your workspace. This is good because you're not making a copy of the JARs for each project, but it's bad because you can't pass this project to another developer via an SCM system or even an exported Eclipse project.

8. Click **Finish** when you've added the required .jar files to create the JSF Implementation library.

Next you may need to fix the contents of the Java Build Path panel (see [Figure 8](#)).

Figure 8. The Java Build Path panel for devSignup



9. Select the **JRE System Library** entry; don't worry if it's for a platform you're not running.
 10. Click the **Edit** button to display the Edit Library dialog box.
 11. Choose a suitable Java system library, then click **Finish**.
 12. Click **OK** in the Properties dialog box to apply your changes. Eclipse rebuild the Java source files, and you should be in business, finally!
-

Section 4. Tomahawk's components

The Tomahawk project loosely divides its components into three categories: components, validators, and other goodies. Let's take a quick look at what each group offers the JSF application developer, with an eye toward extending the existing JSF Developer Forum Signup application that you worked up in the [first tutorial](#) of this series.

One important detail: Many of these components include graphical elements (such as images) or JavaScript™ code that need to be included. They're part of the Tomahawk .jar file and are automatically included in the JSP output courtesy of the MyFaces Extensions Filter. In the next section, you'll learn about that and how to install it so you don't have to worry about missing images and JavaScript code required by the Tomahawk components.

The components

Most of Tomahawk's components, listed in [Table 1](#), are extended or advanced UI components designed to make your Web application look and feel more like a normal desktop application. Some of them provide surprisingly complex widgets, which will save you an enormous amount of time if you happen to need one for your application.

Table 1. Tomahawk components

Component	Description
Alias Bean (Data, <t:aliasBean>)	Lets you create an alias for an existing bean. For example, say you're including an existing subview (<f:subview>) that refers to a bean named <code>holder</code> , but this form's bean is named <code>userData</code> . You can wrap the <f:subview> in a <t:aliasBean> to indicate that any

	references to <code>holder</code> are forwarded to your <code>userData</code> bean.
Buffer (Data, <code><t:buffer></code>)	Renders part of a page into a bean so you can display it later.
Calendar (Input, <code><t:inputCalendar></code>)	Renders a calendar so the user can easily select a date. Week names, month names, and the starting day of the week are automatically chosen from the system locale, so localization is automatic.
Column (Column, <code><t:column></code>)	An extended version of <code><h:column></code> that provides pass-through attributes (that is, they'll be copied into the output HTML directly), letting you precisely control how headers, footers, and data cells are rendered and how they respond to JavaScript events.
Columns (Data, <code><t:columns></code>)	Used inside a Data Table to provide a dynamic count of columns in the table.
Data List (Data, <code><t:dataList></code>)	Similar to the Data Table, but renders its rows as a simple comma-separated list, a bullet list, or a numbered list.
Data Scroller (Panel, <code><t:dataScroller></code>)	A control for flipping through pages of <code>UIData</code> (<code><h:dataTable></code>); this renders first, fast rewind, previous, next, fast forward, and last buttons to control the display.
Date (Input, <code><t:inputDate></code>)	Renders an input text control for entering dates. Can accept date or time (or both), provide a pop-up Calendar for selecting the date, and handle 12-hour or 24-hour times.
Extended Data Table (Data, <code><t:dataTable></code>)	Extends the standard JSF <code>DataTable</code> with the ability to save the state of the <code>DataModel</code> and support for clickable sort headers.
File Upload (Input, <code><t:inputFileUpload></code>)	Renders an HTML input field that lets the user upload a file. Lets you indicate which Multipurpose Internet Mail Extensions (MIME) type is accepted by the field.
Html Editor (Input, <code><t:inputHtml></code>)	Renders a simple inline HTML-based word processor. Currently limited to one HTML editor per page, although you can use tabs to include multiple editors if only one editor is rendered at a time.
Html Tag (Output, <code><t:htmlTag></code>)	Renders any HTML tag.
Javascript Listener (Output, <code><t:jsValueChangeListener></code>)	Replicates the Value Change Listener functionality provided by JavaScript on the client. Use it when a change to one control should trigger changes in other controls. One or more Javascript Listeners can be embedded in any Input control, and they'll all be triggered when that control is updated.

JSCook Menu (Command, <t:jscookMenu>)	Renders a dynamic JavaScript-based menu.
Newspaper Table (Obsolete!)	This is obsolete, so don't use it. The Newspaper Table's functionality is now part of the Extended Data Table component.
Panel Navigation (Panel, <t:panelNavigation>)	Renders a vertical menu with support for nested menu items.
Panel Navigation 2 (Panel, <t:panelNavigation2>)	Similar to the Panel Navigation component, but renders using an unordered list instead of a table. The menu tree can also be built dynamically using Navigation Menu Items.
Panel Stack (Panel, <t:panelStack>)	A stack of Panels, allowing you to switch panels dynamically.
Popup (Panel, <t:popup>)	A pop-up panel that displays on a mouse event (such as when the mouse hovers over another UI component).
Schedule (Panel, <t:schedule>)	Renders an appointment or event schedule in day, work week, week, or month view, similar to the scheduler in the Microsoft® Outlook® client or Evolution.
Style Sheet (Output, <t:stylesheet>)	Renders the path to a common CSS style sheet file.
TabbedPane (Panel, <t:panelTabbedPane>)	Renders a tabbed Panel that lets the user switch between Panels. The switching can be handled by the client or on the server (which causes a page refresh).
Tree (HtmlTree, <t:tree>)	An extended version of the HTML tree component in MyFaces, which lets you specify CSS classes for the tree, the nodes, and the selected node.
Tree2 (HtmlTree, <t:tree2>)	A highly customizable, extended tree component with its data provided by a backing bean. Almost any type of JSF component (text, image, checkbox, and so on) can be rendered inside the tree's nodes, and it supports expansion on the client side (no page refreshes!) in addition to the standard server-side toggles.
Tree Table (HtmlTreeColumn, <t:tree>)	Renders a table with a tree column.
UI Save State (Data, <t:saveState>)	Lets you persist beans and values longer than the Request scope, but shorter than the Session scope. Great for persisting backing beans and values with the same scope as your view components. Traditional JSP applications save their state information in <code>HttpSession</code> objects, which require an expiration date and have several problems related to server restarts and server clusters. The UI Save State component encodes the data in the client response and restores it automatically at the next client

request.

Obviously, some of those components are really useful, while others might be handy in specific situations.

Now take a quick look at Tomahawk's validator components.

The validators

The Tomahawk project also includes several handy validators, listed in [Table 2](#), for ensuring the quality of your input data.

Table 2. Tomahawk validators

Validator	Description
validateCreditCard (<code><t:validateCreditCard></code>)	Validates credit card information for American Express, Discover, MasterCard, or Visa. Can also be used to ensure that the data isn't one of the supported credit cards.
validateEmail (<code><t:validateEmail></code>)	Validates e-mail addresses.
validateEqual (<code><t:validateEqual></code>)	Makes sure that the data in this component matches the data in another component. Handy for verifying passwords or e-mail addresses, for example.
validateRegExpr (<code><t:validateRegExpr></code>)	Ensures that the data in this component matches a specified regular expression. Use this to validate things like all digits in a field, phone numbers, and so on.
validateUrl (<code><t:validateUrl></code>)	Validates a URL.

By reusing these validators, you can save yourself a lot of work. The regular expression validator is especially flexible, assuming you can express the desired data as a regular expression.

Now let's see what other goodies are included in the Tomahawk library.

The other goodies

Tomahawk provides a few other goodies, shown in [Table 3](#), that are helpful (or, in the case of the extensions filter, required if you're using any of the Tomahawk components) for JSF developers.

Table 3. Other Tomahawk goodies

Goody	Description
-------	-------------

Extensions filter	Some of the Tomahawk components require additional supporting JavaScript, CSS style sheets, and images. These resources are included in the Tomahawk .jar file, and the extensions filter provides the code and URLs needed to access those resources from the generated HTML. The extensions filter also handles multipart requests, which are used by the file upload component (<code><t:inputFileUpload></code>).
forceId attribute	This attribute, which can be specified for any of the Tomahawk components, lets you set the ID to the value specified by the ID attribute. This is useful if you need to use JavaScript or CSS styles with HTML element IDs.
JspTilesViewHandler	Tomahawk lets you use tiles from Struts applications, letting you integrate existing tiles with your JSF application.

If you're not already a Struts developer, the extensions filter is probably the most important of these other goodies, since many of the Tomahawk components require it. Since the Tomahawk documentation doesn't seem to include this information, it's best to always use the extensions filter if you want to take advantage of Tomahawk's features. Luckily, it doesn't have much of an impact on performance.

In the next section, you'll look at using Tomahawk with your MyFaces application while developing in Eclipse with the Web Tools Platform.

Section 5. Use Tomahawk with your Web application

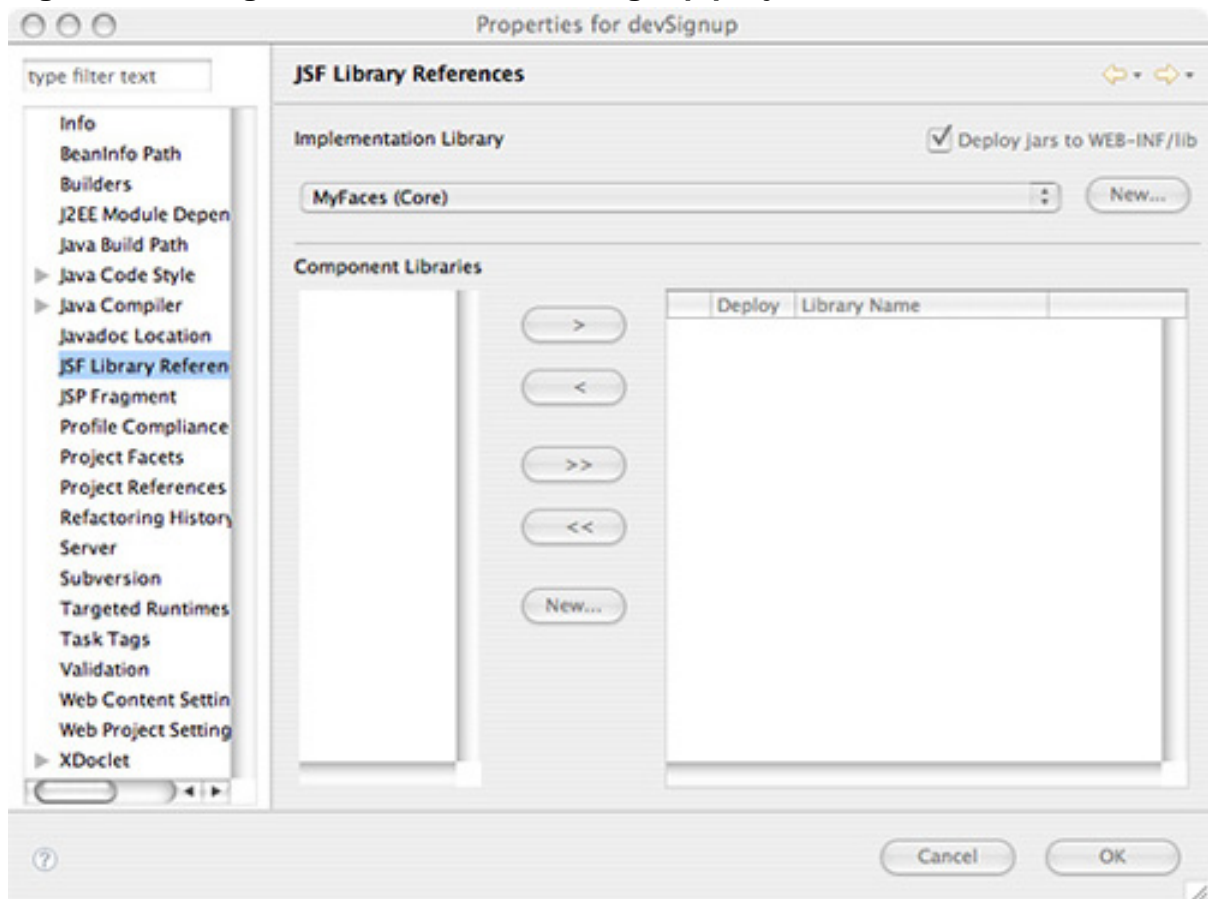
If you're at all like me and you've read this far, you're probably itching to start writing some code and trying out a few of these handy components. First, you should take a detailed look at what you need to do to use Tomahawk with your Web application, and then you can extend the Developer Forum Signup application from the [first tutorial](#) to take advantage of some of the Tomahawk components.

Add Tomahawk to your project

Adding the Tomahawk components to your JSF application is straightforward. You need to add a reference to the Tomahawk .jar file, add the extensions filter to your web.xml file, and include the Tomahawk tag library in your JSF pages.

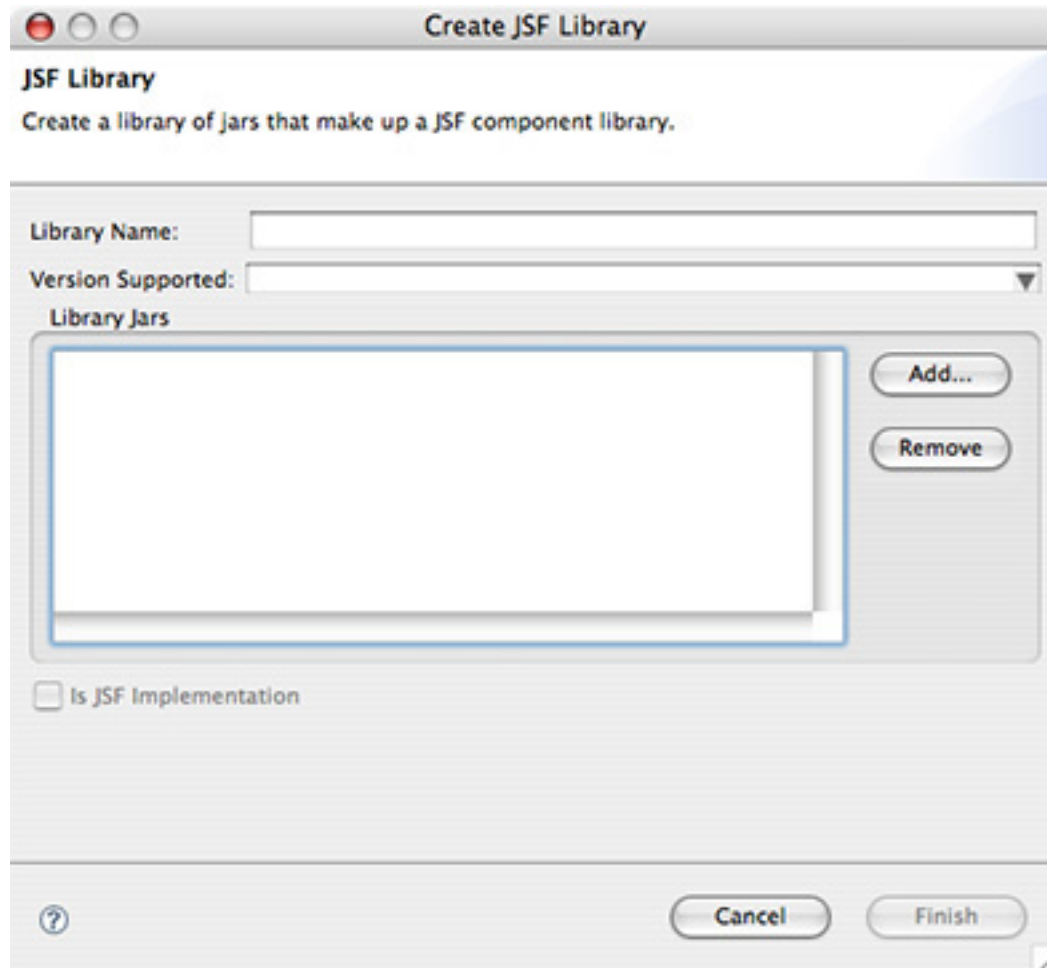
1. Right-click the **devSignup** project in Eclipse's Navigator view, then choose **Properties** from the context menu. Eclipse displays the project properties dialog box.
2. Click **JSF Library References** in the list to display the JSF Library References panel, shown in [Figure 9](#).

Figure 9. Adding Tomahawk to the devSignup project



3. Click the **New** button in the pick-list area (that is, not the one beside the Implementation Library pop-up) to display the Create JSF Library dialog box (see [Figure 10](#)).

Figure 10. Creating a new JSF library



You use this dialog to define a JSF component library, which you can then include in your project. This is definitely handy for adding component libraries that include several JARs, or that depend on additional .jar files from other projects.

4. Enter a library name (I'm using `Tomahawk`) in the Library Name field, then choose `v1_1` from the Version Supported drop-down menu, since MyFaces implements JSF, Version 1.1.
5. Click the **Add** button, then choose the Tomahawk .jar file (`tomahawk-1.1.3.jar`, if you're using the version that's current as of this writing).
6. Repeat this process to add the following Jakarta .jar files, which are required for Tomahawk (there are download links for these in the [Resources](#) section of this tutorial):
 - commons-fileupload-1.1.1.jar
 - commons-validator-1.3.0.jar

- jakarta-oro-2.0.8.jar
7. Click **Finish** to add the new Tomahawk library to the list in the JSF Library References panel of the project properties. The new Tomahawk library is added to the list on the right, which will be automatically included with your project's .war file.
 8. Click **OK** to apply your changes and close the project properties dialog box.
 9. To enable the Tomahawk extensions, you need to add the extensions filter to your Web application. Double-click the project's **web.xml** file (located in the WebContent/WEB-INF folder) to open it in the XML editor. Remember to switch to Source mode by clicking the **Source** tab at the bottom of the editor if it comes up in Design mode -- you want to see raw XML there.
 10. Add the code from [Listing 1](#) to web.xml; the `<filter>` and `<filter-mapping>` elements should be children of the `<web-app>` element, not any of the other web.xml tags.

Listing 1. Enabling the Tomahawk extensions in web.xml

```
<filter>
  <filter-name>MyFacesExtensionsFilter</filter-name>
  <filter-class>
    org.apache.myfaces.webapp.filter.ExtensionsFilter
  </filter-class>

  <init-param>
    <param-name>maxFileSize</param-name>
    <param-value>20m</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>MyFacesExtensionsFilter</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>

<filter-mapping>
  <filter-name>MyFacesExtensionsFilter</filter-name>
  <url-pattern>/faces/myFacesExtensionResource/*</url-pattern>
</filter-mapping>
```

The `<filter>` element adds the MyFaces extensions filter class and sets the `maxFileSize` parameter to 20MB. `maxFileSize` is the maximum size allowed by the file upload component and is specified in bytes (a number), kilobytes (a number followed by `k`), megabytes (a number followed by `m`), or gigabytes (a number followed by `g`).

The first `<filter-mapping>` element applies the MyFaces extensions filter to the

specified servlet. The `<servlet-name>` must match the servlet name specified in the `<servlet>` element that's already in your web.xml file.

The second `<filter-mapping>` element tells the MyFaces extensions filter which URLs to match when handling the additional JavaScript, style sheets, and images used by the Tomahawk components. URLs matching this pattern are intercepted and translated into references to the appropriate resources in the Tomahawk .jar file.

The last thing you need to do is enable the Tomahawk extensions in your .jsp files.

11. Add the code below to every JSP page that'll be using Tomahawk components. In the devSignup project, that's signup.jsp (the response pages aren't using any JSF code).

```
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"
%>
```

That's all there is to it! Now you're ready to use some of the Tomahawk components in your sample application.

Add to the Developer Forum Signup application

The original Developer Forum Signup application, which you created in the first tutorial, is very basic. It lets the user enter a screen name, an e-mail address, and a password -- that's about it. No data is validated, and there's no mechanism for making sure the user entered the password they think they entered. [Listing 2](#) shows you the current state of the main signup page, signup.jsp (note that I've already added the Tomahawk tag library just before the `<f:view>` element).

Listing 2. The Developer Forum Signup page

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t" %>
<f:view>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Developer Forum Signup</title>
</head>
<body>
<h1>Developer Forum Signup</h1>
<p>
Welcome to our forums! Please fill in the following form to create your
forum account.
```

```

</p>
<h:form>
<dl>
  <dt>Screen name:</dt>
  <dd>
    <h:inputText value="#{signupData.screenName}"/>
  </dd>

  <dt>Email:</dt>
  <dd>
    <h:inputText value="#{signupData.email}"/>
  </dd>

  <dt>Password:</dt>
  <dd>
    <h:inputSecret value="#{signupData.password}"/>
  </dd>
</dl>

<h:commandButton action="#{signupData.register}">Sign
  up</h:commandButton>
</h:form>
</body>
</html>
</f:view>

```

It's a trivial page, using MyFaces components for gathering the user's data and performing the `signupData` bean's `register()` method when the **Sign up** button is clicked.

The first thing you should add is a validator to make sure the user has entered a plausible e-mail address.

1. Edit `signup.jsp` in your `devSignup` project (remember to add the Tomahawk tag library statement if you haven't already!) and replace the Email field's `<h:input>` tag with the code in [Listing 3](#).

Listing 3. Validating the user's e-mail address.

```

<h:inputText value="#{signupData.email}" id="email" required="true">
  <t:validateEmail/>
</h:inputText><br/>
<strong><h:message for="email"/></strong>

```

That's all you need to do. Tomahawk will automatically check the entered data to make sure it's a valid e-mail address. If it's not, it will set the message for the ID `email` to an appropriate error message. The `<h:message>` element will display the message for ID `email` if there's a message (otherwise, it displays nothing, so you'll only see this message if an error occurred).

For example, if you entered `not!email` as the e-mail address, you might see something like [Figure 11](#) after clicking the "Sign up" button.

Figure 11. Entering an invalid e-mail address

Developer Forum Signup

Welcome to our forums! Please fill in the following form to create your forum account.

Screen name:

Email:

The given value (not!email) is not a correct email-address.

Password:



The form is displayed again. This time, the message for the Email field is set, so it's displayed and... Oh look, it's an error message -- exactly what was expected.

Now you should require the user to enter a password twice; because the user can't see what he or she is entering, this will help the user verify that the expected password was entered correctly.

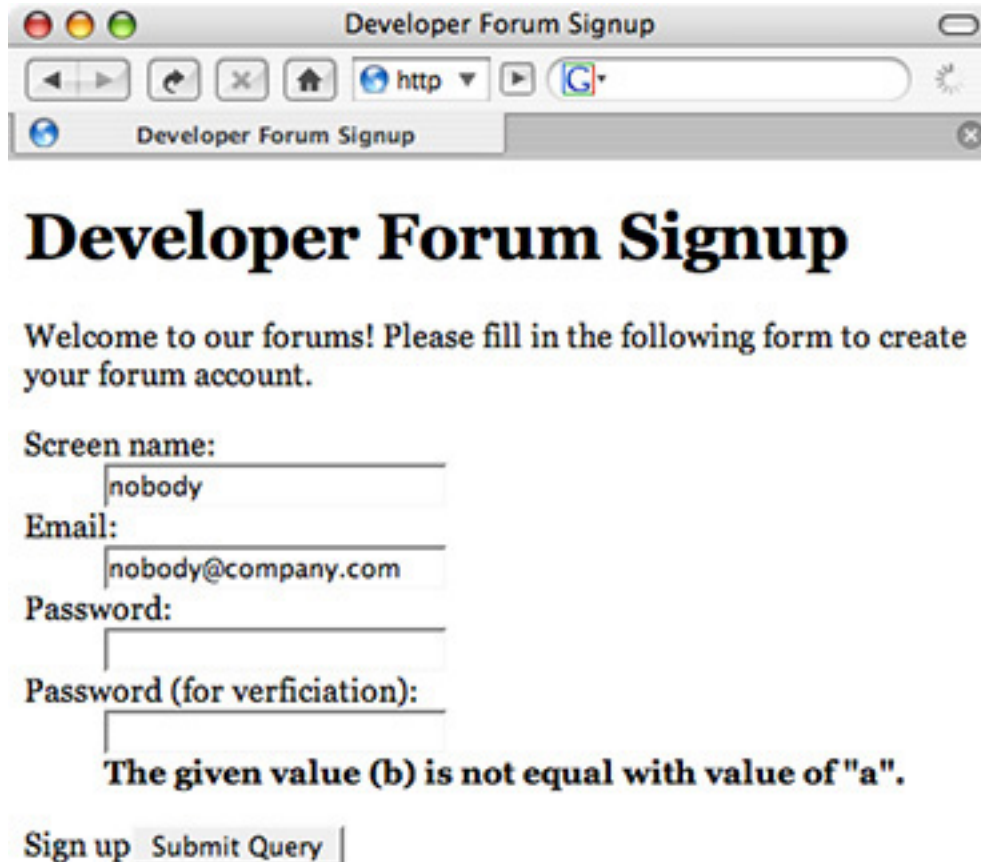
2. Add the code from [Listing 4](#) after the `</dd>` tag for the Password field. This adds a second Password entry field, and whatever the user enters here must match the contents of the first Password field.

Listing 4. Double password entry

```
<dt>Password (for verification):</dt>
<dd>
<h:inputSecret id="password_verify" required="true">
  <t:validateEqual for="password"/>
</h:inputSecret><br/>
<strong><h:message for="password_verify"/></strong>
</dd>
```

This uses `<t:validateEqual>` to make sure the contents are the same, and `<h:message>` to display any messages associated with this part of the form. You can see what its error messages look like in [Figure 12](#).

Figure 12. You entered two passwords that don't match



The screenshot shows a web browser window titled "Developer Forum Signup". The address bar shows "http://". The page content includes a heading "Developer Forum Signup" and a welcome message: "Welcome to our forums! Please fill in the following form to create your forum account." The form has four fields: "Screen name:" with the value "nobody", "Email:" with the value "nobody@company.com", "Password:", and "Password (for verification):". Below the form, a red error message reads: "The given value (b) is not equal with value of 'a'". At the bottom of the form, there are two buttons: "Sign up" and "Submit Query".

As you can see, this validator isn't secure. At least you haven't had to change your Java code, even though you've added some good functionality to the signup page.

How about adding something a little flashier? Ask the person signing up for his or her birthday. You can use this to make sure the user is over 18 (if that's an issue), and you can do "cute" things like greet the user with birthday wishes on the appropriate day.

3. Add the code from [Listing 5](#) to your `signup.jsp` file, just before the `</dl>` tag.

Listing 5. Adding a date input control

```
<dt>Birthday:</dt>
<dd>
<t:inputDate id="birthday" value="#{signupData.birthday}" popupCalendar="true"/>
</dd>
```

Figure 13 shows a birthday prompt and a date input control with support for a pop-up calendar so the user can pick his or her birthday instead of entering it by hand.

Figure 13. Adding a birthday prompt and a date input control



Since you're referring to a date member of the `signupData` bean, you're also going to have to (finally) modify some Java code.

4. Edit the `SignupData.java` file (in your `devSignup` project, under `src/devSignup`), and add the code from [Listing 6](#) to your `SignupData` class.

Listing 6. Adding birthday support to the bean.

```
        private java.util.Date _birthday = null;

    public SignupData() {
        _birthday = new java.util.Date();
    }

    public java.util.Date getBirthday() {
        return _birthday;
    }

    public void setBirthday( java.util.Date newBirthday ) {
        this._birthday = newBirthday;
    }
}
```

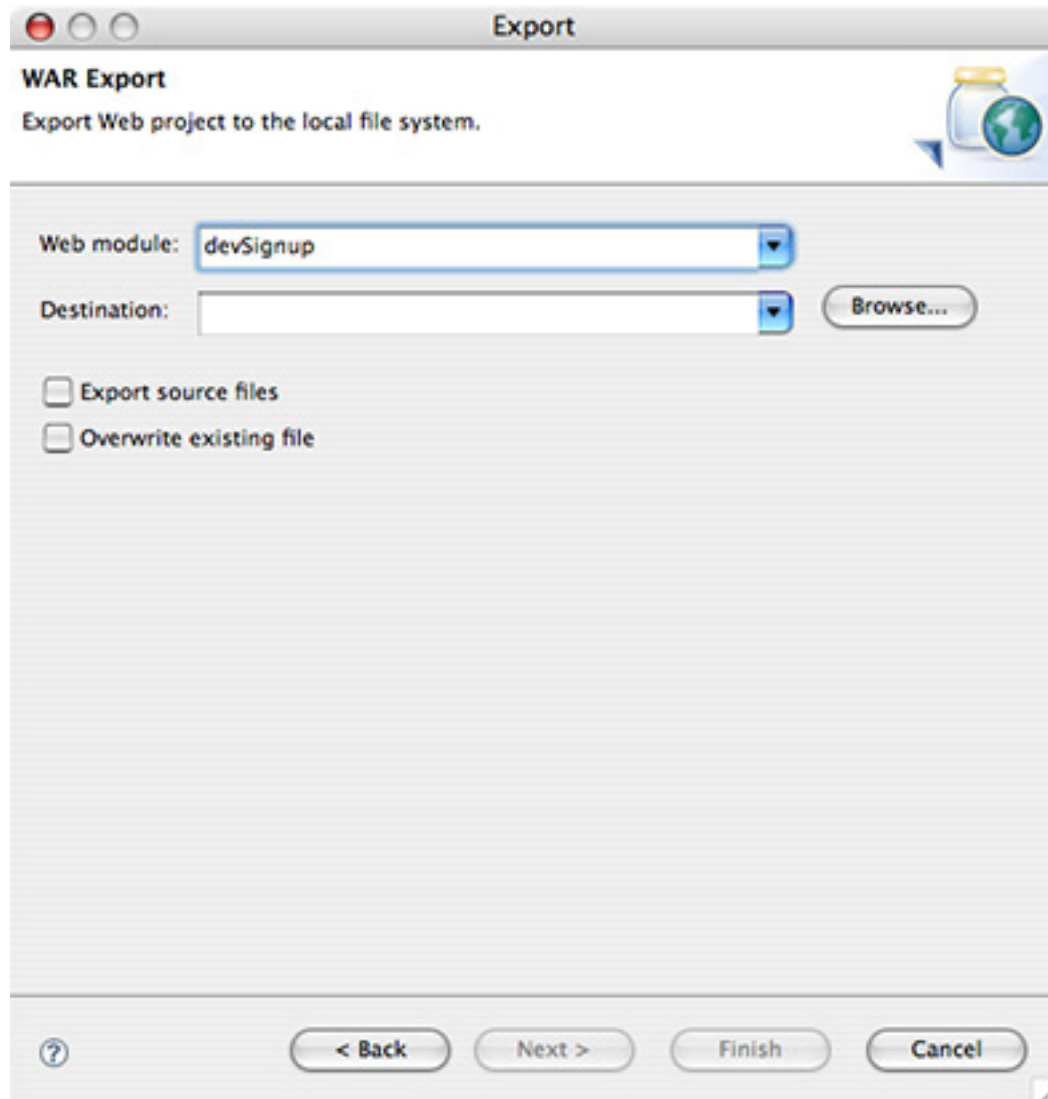
Not much work to add a very complex data-entry widget to the Web page! Now you can build and deploy the Web application to see what you've done.

Deploy the new Developer Signup application

You've changed a few things, and now is a good time to see how things are working out with your Developer Signup application. You need to build the project, export a .war file, and deploy it to the Geronimo server.

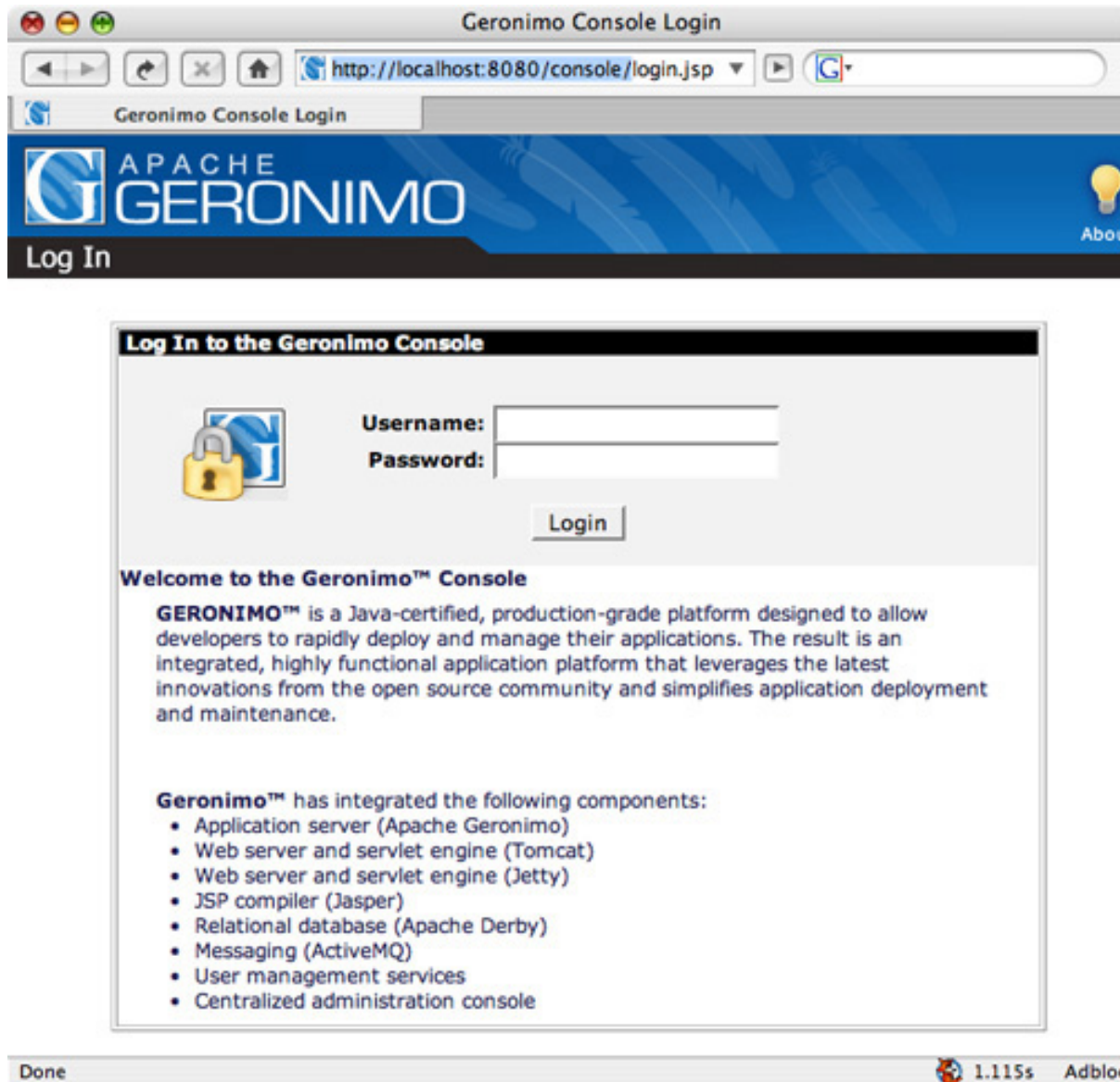
1. Click your **devSignup** project in Eclipse's Navigator view, then choose **Project > Build Project** from the menu. Eclipse compiles your Java code and validates your JSP pages and XML files.
2. Right-click **devSignup** in the Navigator view, then choose **Export** from the context menu.
3. In the Export wizard, expand the Web group, choose **WAR file**, then click **Next** to display the WAR Export panel, as shown in [Figure 14](#).

Figure 14. Exporting a .war file



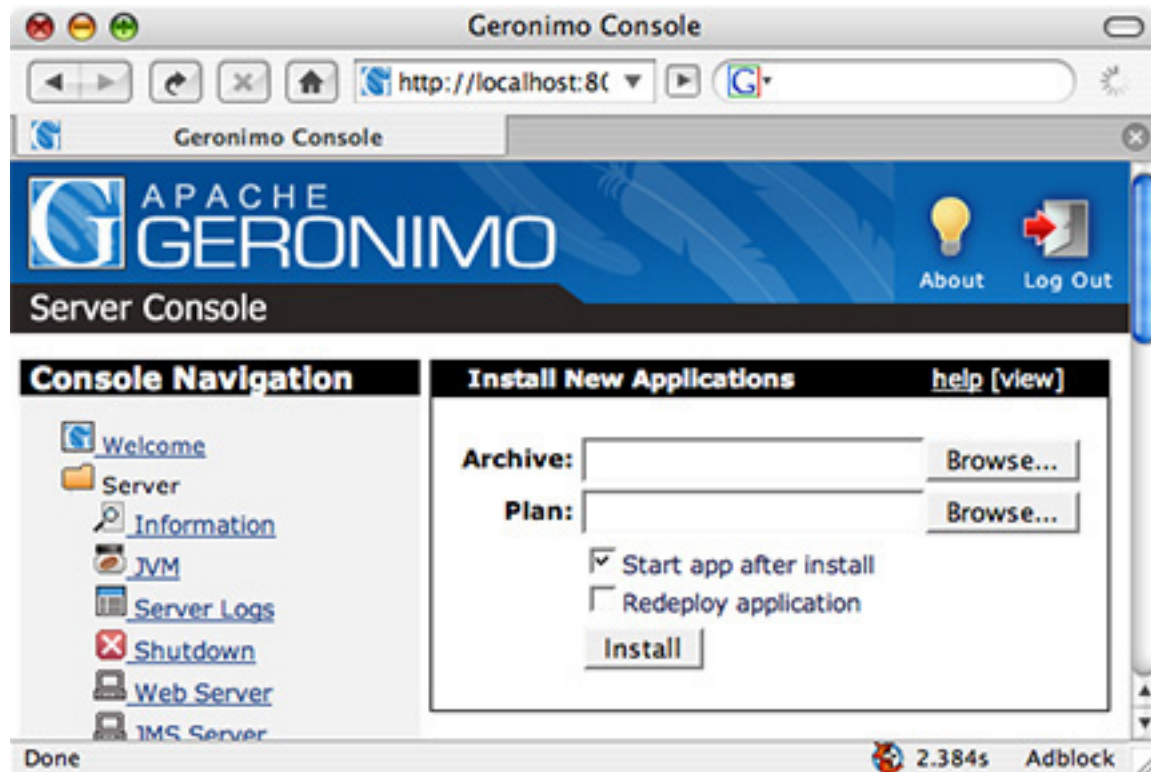
4. Click the **Browse** button to select a destination directory and file name for your .war file. (I'm saving mine to my desktop as devSignup.war so I can easily find it when I go to deploy it on the server.)
5. Click **Finish** to export the .war file.
6. Fire up your favorite Web browser and access `http://localhost:8080/console/` (substitute your Geronimo server's host name for localhost if it's not running on the same machine). This displays the Geronimo Console Login screen (see [Figure 15](#)).

Figure 15. Logging in to Geronimo's management console



7. Log in to access the administration console. (Remember, the default user name is *server*, and the default password is *manager* -- you should change these.)
8. Click **Deploy New** from the Applications section in the Console Navigation list to display the Install New Applications screen, shown in [Figure 16](#).

Figure 16. Installing a new Web application



9. Click the **Browse** button next to the Archive field, and browse to your recently created .war file.
10. Because **Start app after install** is already checked, click **Install** to upload the .war file and launch your application. You'll see a success message after your browser finishes uploading the archive and Geronimo finishes launching your application.
11. If you already have devSignup running on your server because you followed along with the first tutorial, check the **Redeploy application** box to install the new version and uninstall the old version.

Now when you access the Developer Signup application (which is `http://localhost:8080/devSignup/signup.faces` on my system, but use your server's host name instead of `localhost` if the server isn't running on your local machine), you should see something like [Figure 17](#).

Figure 17. The improved application



The screenshot shows a web browser window with the title "Developer Forum Signup". The address bar contains "Developer Forum Signup". The main content area displays the following form:

Developer Forum Signup

Welcome to our forums! Please fill in the following form to create your forum account.

Screen name:

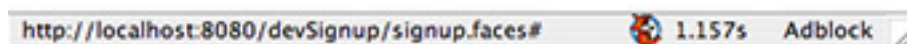
Email:

Password:

Password (for verification):

Birthday:
 ...

Sign up



When the user clicks the ... button beside the Birthday entry field, something like [Figure 18](#) will appear. (I couldn't resist showing this because it's very complex, but we didn't have to do anything -- it's supported automatically!)

Figure 18. The date input field's pop-up calendar



That's pretty awesome, and it's done entirely on the client's system with JavaScript and CSS. The only drawback is that it looks nothing like a "native" application.

That's it! You've successfully explored Apache Tomahawk and used it to extend the Developer Signup Web application.

Section 6. Summary

You've examined the Apache Tomahawk components, discovered how to add them to your Eclipse Web application project, and used them to extend the Developer Forum Signup application you started in the first tutorial.

In the next installment of this series, Part 3, you'll look at adding Ajax capabilities to your JSF application and use them to make the Developer Forum Signup application more interactive without requiring more page reloads or transitions.

Downloads

Description	Name	Size	Download method
Part 2 source code	devSignup-src.zip	11KB	HTTP
Signup2 application WAR file	devSignup-war.zip	3160KB	HTTP

[Information about download methods](#)

Resources

Learn

- Read "[JSF KickStart: A Simple JavaServer Faces Application](#)," a tutorial that shows you an example of a JSF application developed without any special IDE.
- Check out the JSF tutorial series, [A Quick Introduction to JavaServer Faces Plus Apache MyFaces Extensions](#), which includes downloadable JSF applications for testing on your own server.
- See the [Web Tool Platform 1.5's list of JSF features](#) (JSF support is 0.5 at this point).
- Check out the [Sun JSF tag reference documentation](#) for the h: and f: tags.
- Read the article "[Deploy J2EE applications on Apache Geronimo](#)" (developerWorks, January 2006).
- Read the tutorial "[Using regular expressions](#)" (developerWorks, September 2000) for an introduction to using regular expressions.
- Join the [Apache Geronimo mailing list](#).
- Check out the developerWorks [Apache Geronimo project area](#) for articles, tutorials, and other resources to help you get started developing with Geronimo today.
- Find helpful resources for beginners and experienced users at the [Get started now with Apache Geronimo](#) section of developerWorks.
- Check out the [IBM Support for Apache Geronimo](#) offering, which lets you develop Geronimo applications backed by world-class IBM support.
- Visit the [developerWorks Open source zone](#) for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Browse all the [Apache articles](#) and [free Apache tutorials](#) available in the developerWorks Open source zone.
- Browse for books on these and other technical topics at the [Safari bookstore](#).

Get products and technologies

- Visit the [Apache MyFaces Project](#) Web site.
- Get the following Apache goodies:
 - [Geronimo J2EE application server](#)
 - [MyFaces JSF implementation](#)

- [Tomahawk JSF components for MyFaces](#)
- [Sandbox](#), an incubator for Tomahawk components
- Get the following Tomahawk dependencies:
 - [Commons FileUpload](#)
 - [Commons Validator](#)
 - [Jakarta ORO](#)
- Get the [JSR 127: The JavaServer Faces specification](#) straight from Sun Microsystems.
- Download [Eclipse](#).
- Download the [Eclipse Web Tools Platform](#).
- Innovate your next open source development project with [IBM trial software](#), available for download or on DVD.
- Download your free copy of [IBM WebSphere® Application Server Community Edition V1.0](#) -- a lightweight J2EE application server built on Apache Geronimo open source technology that is designed to help you accelerate your development and deployment efforts.

Discuss

- [Participate in the discussion forum for this content.](#)
- Stay up to date on Geronimo developments at the [Apache Geronimo blog](#).
- Get involved in the developerWorks community by participating in [developerWorks blogs](#).

About the author

Chris Herborth



Chris Herborth is an award-winning senior technical writer with more than 10 years of experience writing about operating systems and programming. When he's not playing with his son Alex or hanging out with his wife Lynette, Chris spends his spare time designing, writing, and researching (that is, playing) video games.

Trademarks

IBM, the IBM logo, and WebSphere are registered trademarks of IBM in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Outlook are trademarks of Microsoft Corporation in the United States, other countries, or both.