

# Build Apache Geronimo applications using JavaServer Faces, Part 5: Integrating your JSF application with Spring

Skill Level: Intermediate

[Chris Herborth](#)  
Freelance  
Freelance Writer

19 Dec 2006

This final installment wraps up the five-part tutorial series by introducing you to the Spring Framework. Learn to integrate your Java™Server Faces (JSF) applications with Spring, a popular framework that makes it easier for Apache Geronimo developers to build Java Platform, Enterprise Edition (Java EE) applications. You'll use Spring to continue developing the front end for the example Developer Forum Signup application.

## 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 the Spring Framework, a large (and very complete) Web application framework that you can incorporate with your JSF application. You'll make use of Spring to continue developing the front end for the signup pages of our developer forum example application.

## About this series

This is the final tutorial of a five-part series about building Apache Geronimo applications using JSF. The tutorials in the series include:

- **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** showed 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** demonstrated 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** taught 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.
- [Spring Framework](#), a powerful application framework that can be integrated with existing Web applications.
- [Eclipse](#), the extensible open source IDE that supports a wide range of languages and platforms.

- [Eclipse Web Tools Platform](#), 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). I'll be using Java 1.5 on Mac OS X 10.4, but it shouldn't make much of a difference.
- 

## Section 2. Getting ready

In this section, you'll import the current devSignup project so you can start mixing Spring functionality into the application in the next section. You'll also learn about the Spring Framework and some of the advantages it provides.

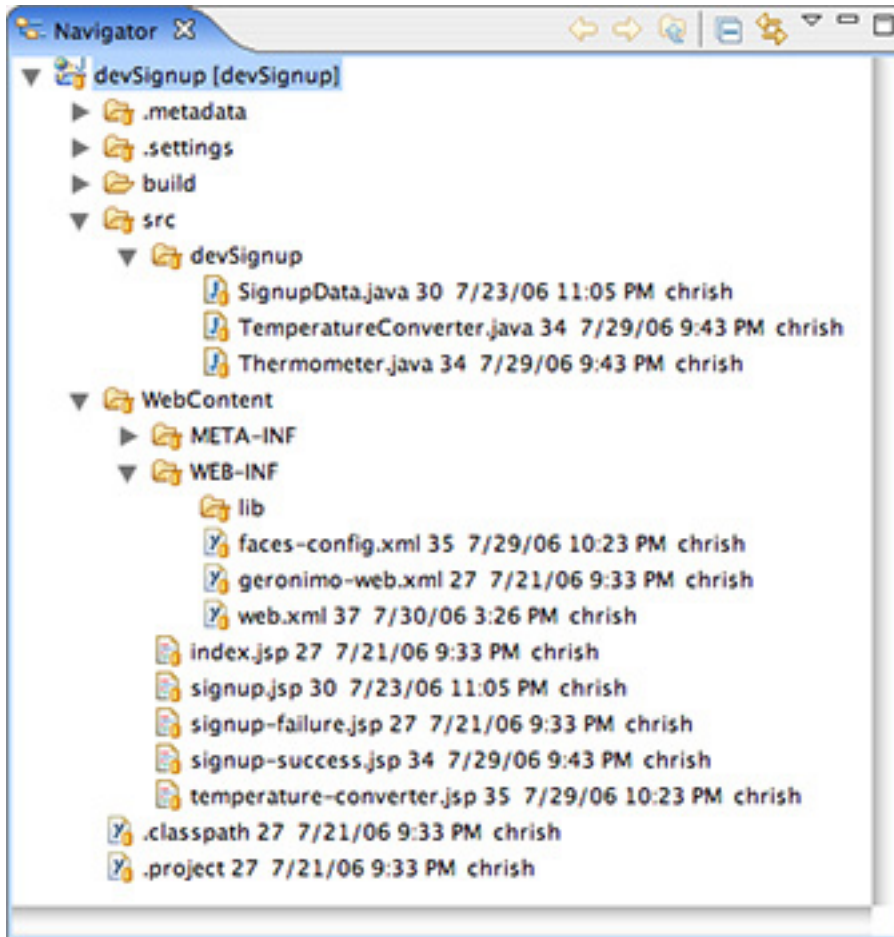
### Import the devSignup project

If you haven't read the previous tutorials in this series, you should at least download the devSignup sample project (linked in the [Downloads](#) section at the end of this tutorial), because you'll be adding to it later.

[Part 2](#) has detailed instructions for importing the project into Eclipse from the source archive. Download devSignup-src.zip, and follow those instructions from Part 2, then come back here when you're done. Be sure to follow the instructions in the "Fix the devSignup project" section of that tutorial, too, or you won't be able to build the application.

When you're done, your Eclipse Navigator view should look something like the one shown in [Figure 1](#).

#### **Figure 1. The devSignup project in Eclipse**



Now you can look at the Spring Framework and start thinking about what it can do for you.

## The Spring Framework

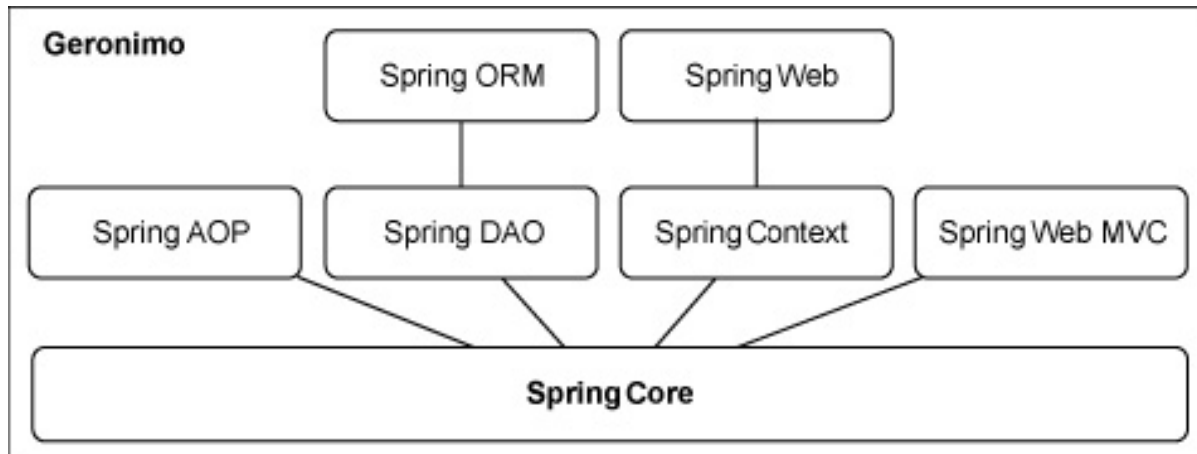
Spring is a large Java Web application framework designed to make Java programming less painful. It integrates well with existing tools like Hibernate, and it hides the differences between different Java Database Connectivity (JDBC) implementations. It offers a layered design (see [Figure 2](#)) and sports several helpful features, including the following:

- A complete lightweight container for your application and its objects
- A common abstraction layer for transaction management, which this isn't tied to any particular Java environment and hides the low-level details (and issues) from the developer
- A JDBC abstraction layer, providing meaningful and standardized exception behavior across JDBC implementations

- Integration with various resource holders, including TopLink, Hibernate, Java Data Objects (JDO), and iBATIS SQL Maps
- Aspect-Oriented Programming (AOP), integrated into the Spring configuration management
- A flexible Model View Controller (MVC) Web application framework

You can see Spring's major modules in [Figure 2](#).

**Figure 2. Spring's architecture filled to the brim with Java EE acronyms**



In [Figure 2](#) you can see Spring running inside of a Geronimo application server, which gives you a general idea about its layered design. Spring lets you inject objects using JavaBean properties and configuration files; this lets you dynamically add or remove collaborating objects and services. For developers familiar with the Decorator design pattern, this is similar, but without requiring any code rebuilding; it can all be done through the configuration files.

There are a lot of high-end, enterprise application features and support in Spring that you're going to ignore in this tutorial so you can focus on seeing how to take advantage of the Spring Framework in your existing application.

---

## Section 3. Reviewing the MyFaces JSF implementation of devSignup

Now that you've learned a bit more about Spring, you should look at some actual code.

First you'll take a look at what you've done to make the Developer Forum Signup

application run within Geronimo using the Apache MyFaces JSF implementation. Then in the next section you'll modify it so you can add some Spring functionality.

## MyFaces Developer Signup application

Currently, the Developer Signup application is fairly simple. It presents one page with a few fields the user can fill in, validates some of the data (warning the user if invalid data is detected), and tells the user if they've succeeded or not. In fact, I've gone back to the version from [Part 2](#) in this series ("Using Tomahawk with Java Server Faces"), so it's just using plain JSF (in the form of Apache MyFaces) with a few extensions from the Tomahawk component library. This gives us a nice bit of functionality, without much complexity.

The most important files in this small application are:

- web.xml -- Java EE application descriptor, settings, and so on
- faces-config.xml -- JSF configuration
- geronimo-web.xml -- Geronimo application descriptor
- signup.jsp -- The signup page and validation
- SignupData.java -- The bean that keeps track of the form data

The source code for the base JSF version of this application is available in the [Downloads](#) section (as devSignup-jsf-src.zip) if you want to load it into Eclipse and experiment with the whole thing. You'll take a quick look at the important source files now.

### devSignup web.xml

[Listing 1](#) shows you the complete web.xml for the Developer Signup application.

#### Listing 1. Developer Signup's web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="devSignup" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>devSignup</display-name>

  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
  </context-param>
```

```

    <context-param>
<param-name>org.apache.myfaces.ALLOW_JAVASCRIPT</param-name>
  <param-value>>true</param-value>
</context-param>

    <context-param>
  <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
  <param-value>>true</param-value>
</context-param>

    <context-param>
<param-name>org.apache.myfaces.DETECT_JAVASCRIPT</param-name>
  <param-value>>false</param-value>
</context-param>

    <context-param>
  <param-name>org.apache.myfaces.AUTO_SCROLL</param-name>
  <param-value>>true</param-value>
</context-param>

    <context-param>
<param-name>org.apache.myfaces.ADD_RESOURCE_CLASS</param-name>
<param-value>org.apache.myfaces.renderkit.html.util.DefaultAddResource
</param-value>
  </context-param>

    <context-param>
<param-name>org.apache.myfaces.CHECK_EXTENSIONS_FILTER</param-name>
>
  <param-value>>true</param-value>
</context-param>

    <context-param>
<param-name>org.apache.myfaces.ADD_RESOURCE_CLASS</param-name>
<param-value>org.apache.myfaces.renderkit.html.util.DefaultAddResource
</param-value>
  </context-param>

    <context-param>
<param-name>org.apache.myfaces.CHECK_EXTENSIONS_FILTER</param-name>
>
  <param-value>>true</param-value>
</context-param>

    <listener>
<listener-class>org.apache.myfaces.webapp.StartupServletContextListener
</listener-class>
  </listener>

    <servlet>
  <servlet-name>Faces Servlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

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

```

```

        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>

    <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>
        <url-pattern>*.faces</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>MyFacesExtensionsFilter</filter-name>

    <url-pattern>/faces/myFacesExtensionResource/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>

```

This web.xml file contains several groups of elements, listed in [Table 1](#).

**Table 1. Elements in web.xml file**

Elements	Descriptions
<web-app>	Container for the Web application descriptor data
<display-name>	Application name, as displayed to the system administrator
<context-param>	Key and value pairs that set various configuration values used by the Web application components; most of the ones shown here are defaults, but are specified anyway to remove warnings from the application startup logs
<listener>	Specifies the name of the class that will listen for incoming client connections and decide how to handle them
<servlet>	Where you specify the internal name of the servlet, the name of the class that manages the servlet's components, and whether the servlet should be loaded when the application server starts
<servlet-mapping>	Defines the URLs that will be fed through the matching <servlet>, as specified by the <servlet-name>, which has to match the <servlet-name> specified in the <servlet> section

<filter> and <filter-mapping>	Adds a page filter to the application and indicates which URLs need to be passed through the filter, like the <servlet> and <servlet-mapping> pair
<welcome-file-list>	List of one or more file names that should be loaded if an incomplete URL is passed to the application

These are all standard Java EE application descriptor tags; there's nothing specific to JSF, MyFaces, or Tomahawk there.

## devSignup faces-config.xml

The faces-config.xml file (see [Listing 2](#)) sets JSF application parameters and declares managed beans and the navigation rules that link one JSF page to the next.

### Listing 2. JSF application parameters are in faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <application>
    <locale-config>
      <default-locale>en_US</default-locale>
    </locale-config>
  </application>

  <managed-bean>
    <description>
      Data used for requesting membership in the Developer Forum.
    </description>
    <managed-bean-name>signupData</managed-bean-name>

<managed-bean-class>devSignup.SignupData</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>

  <navigation-rule>
    <display-name>signup</display-name>
    <from-view-id>/signup.jsp</from-view-id>
    <navigation-case>
      <from-outcome>signup-success</from-outcome>
      <to-view-id>/signup-success.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
  <navigation-rule>
    <display-name>signup</display-name>
    <from-view-id>/signup.jsp</from-view-id>
    <navigation-case>
      <from-outcome>signup-failure</from-outcome>
      <to-view-id>/signup-failure.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

```
</faces-config>
```

This file's elements, listed in [Table 2](#), are slightly more complex than what you saw in `web.xml`, but you created most of them through Eclipse's Web Tooling Platform's graphical editors.

**Table 2. Elements in `faces-config.xml` file**

Elements	Descriptions
<code>&lt;faces-config&gt;</code>	Container for JSF application data
<code>&lt;application&gt;</code>	Lets you specify application settings, supported languages, and so on
<code>&lt;managed-bean&gt;</code>	Each <code>&lt;managed-bean&gt;</code> tag defines and describes a managed bean used by the application, including the name it's referred to in JavaServer Pages (JSP) pages, the name of the Java class that implements the bean, and the bean's scope.
<code>&lt;navigation-rule&gt;</code>	Each <code>&lt;navigation-rule&gt;</code> indicates a source page, a response string (this comes from a method in the bean, called from the source page), and a destination page for that response.

I find the `<navigation-rule>` blocks a little verbose; I would have allowed multiple `<navigation-case>` elements in each `<navigation-rule>`, which would make things a little cleaner as far as I'm concerned, but I didn't design JSF.

## devSignup `geronimo-web.xml`

The `geronimo-web.xml` file is fairly trivial (see [Listing 3](#)) and indicates the URL where the application is exposed on the server.

**Listing 3. Where Geronimo puts the application (`geronimo-web.xml`)**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
  xmlns:naming="http://geronimo.apache.org/xml/ns/naming">
  <context-root>/devSignup</context-root>
</web-app>
```

Nothing to it, right?

## devSignup `signup.jsp`

As you can see from [Listing 4](#), the Developer Signup entry form is fairly straightforward, using text input fields and some basic Tomahawk validator components.

#### Listing 4. The Developer Signup entry form

```

<?xml version="1.0" encoding="UTF-8" ?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:t="http://myfaces.apache.org/tomahawk">
  <jsp:output omit-xml-declaration="false" doctype-root-element="html"

  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
  doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" />
  <jsp:directive.page contentType="text/html; charset=utf-8" />
  <f:view>
    <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}"
              id="email" required="true">
              <t:validateEmail />
            </h:inputText><br />
            <strong><h:message for="email"
          /></strong></dd>

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

            <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>

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

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

```

```
</f:view>
</jsp:root>
```

This is a little different from the `signup.jsp` that was included in [Part 2](#) of this series; this version is using the XML syntax instead of the JSP syntax. (I love well-formed XML!)

## devSignup SignupData.java

Finally, here's the source for the managed bean you use in this application (see [Listing 5](#)).

### Listing 5. The Developer Signup application's managed bean

```
package devSignup;

public class SignupData {
    // Properties and accessors.
    private String _screenName = "nobody";

    private String _email = "nobody@company.com";

    private String _password = "";

    private java.util.Date _birthday = null;

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

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

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

    public String getScreenName() {
        return this._screenName;
    }

    public void setScreenName(String newScreenName) {
        this._screenName = newScreenName;
    }

    public String getEmail() {
        return this._email;
    }

    public void setEmail(String newEmail) {
        this._email = newEmail;
    }

    // NOTE: THIS IS NOT SECURE, DON'T DO THIS
    // WITH YOUR PASSWORDS!
    public String getPassword() {
        return this._password;
    }
}
```

```
public void setPassword(String newPassword) {
    this._password = newPassword;
}

// The registration attempt handler.
public String register() {
    if ((this._email == null) || (this._screenName == null)
        || (this._password == null)) {
        // Bad code? lost data?
        return "signup-failure";
    }

    if ((this._email.trim().length() < 3)
        || (this._email.indexOf("@") == -1)) {
        // Bad email address.
        return "signup-failure";
    }

    if (this._password.trim().length() < 8) {
        // Password too short.
        return "signup-failure";
    }

    return "signup-success";
}
}
```

This is a simple bean that manages all the data you're collecting in the form. It also provides one method, `register()`, which you call from the form when the user clicks the **Sign up** button. This returns one of the response strings that you saw earlier in the `faces-config.xml` file. These response strings are used by the JSF engine to figure out what page to display next.

Now that I've dragged you back through the MyFaces version of the Developer Signup application, you should figure out how to add in Spring functionality.

---

## Section 4. Adding Spring MVC to devSignup

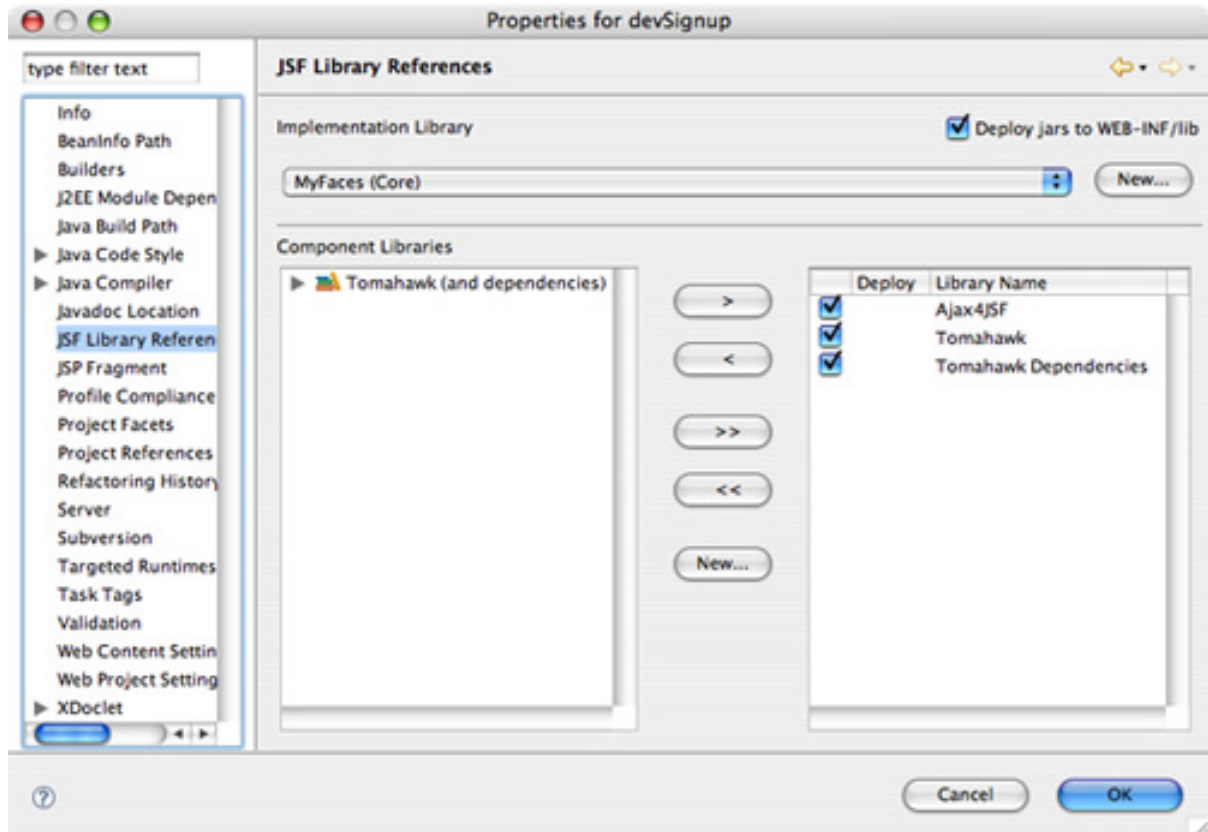
You now have devSignup ready to add in Spring MVC functionality, so let's jump right in.

### Get started

1. Right-click the **devSignup** project in Eclipse's Navigator view, then choose **Properties** from the context menu. Eclipse displays the Properties dialog box.

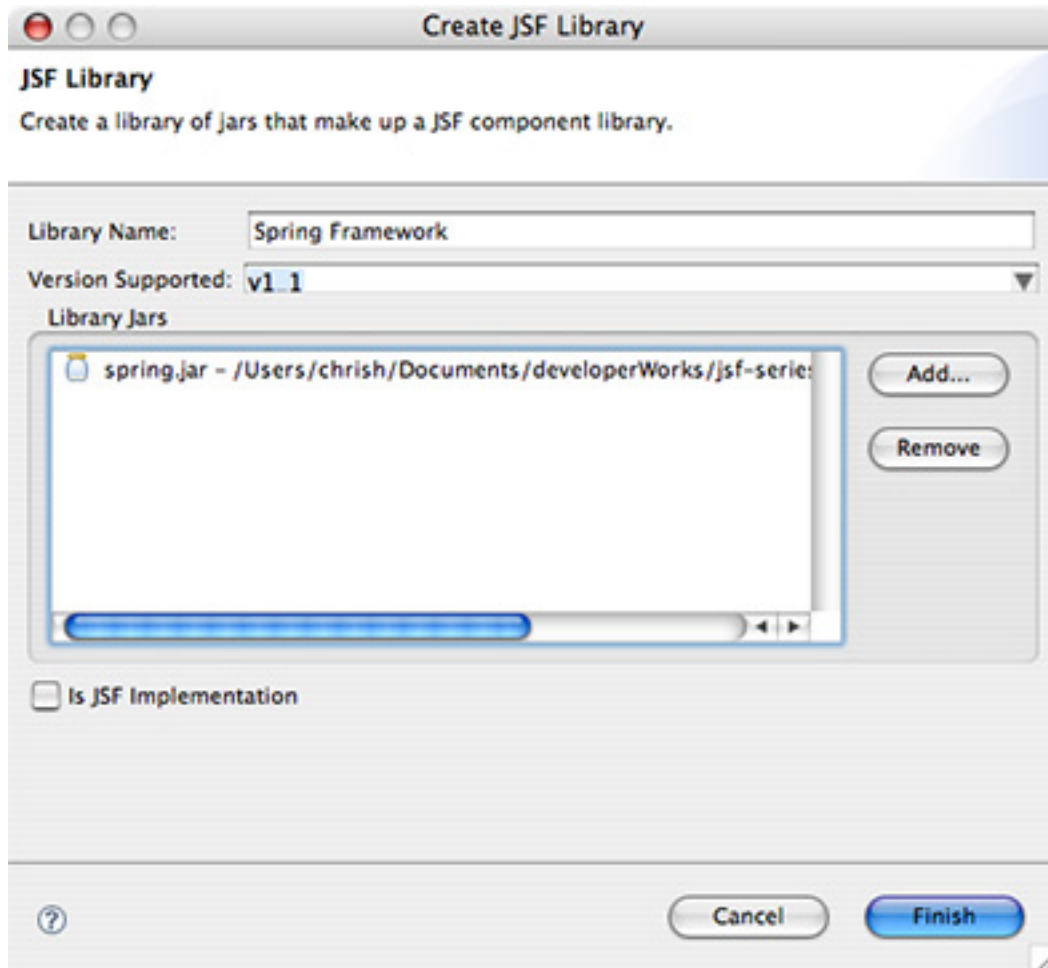
2. Click **JSF Library References** in the list to display the JSF Library References panel (see [Figure 3](#)).

**Figure 3. Adding Spring to the devSignup project**



3. Click the **New** button in the pick-list area (that is, not the one beside the Implementation Library drop-down list) to display the Create JSF Library dialog box. You use this dialog to define a JSF component library, which you can then include in your project. This is handy for adding component libraries that include several JAR files, or that depend on additional JAR files from other projects.
4. Enter a library name (I'm going to use `Spring Framework`) in the **Library Name** field, then choose **v1\_1** from the Version Supported drop-down menu, since MyFaces implements JSF 1.1.
5. Click the **Add** button, then choose the Spring .jar file: `spring.jar` (see [Figure 4](#)).

**Figure 4. Making the Spring Framework library reference**



Yes, that's it. The Spring Framework also ships with various other JAR files; the `spring.jar` contains everything found in these. If you're after finer-grained control of your application's contents or you're pressed for space on the server, you can pick and choose from these. Refer to the `readme.txt` file that comes with the Spring distribution for a complete listing of each JAR file's dependencies. There's one more file (`spring-mock.jar`), which includes mock server contexts and other classes that make unit testing a breeze:

- `spring-core.jar` -- Core utilities; everything requires this
- `spring-beans.jar` -- JavaBeans support, bean container
- `spring-aop.jar` -- AOP framework, source-level metadata support, AOP Alliance interfaces
- `spring-context.jar` -- Application context, validation, Java Naming and Directory Interface (JNDI), UI context support
- `spring-dao.jar` -- Data Access Objects (DAO) support, transaction infrastructure

- spring-jdbc.jar -- JDBC support
  - spring-support.jar -- Java Management Extensions (JMX) support, J2EE Connector Architecture (JCA) support, scheduling support, mail support, caching support
  - spring-web.jar -- Web application context, multipart resolver, Struts support, JSF support, Web utilities
  - spring-webmvc.jar -- Framework servlets, Web MVC framework, Web controllers, Web views
  - spring-remoting.jar -- Remoting support, Enterprise JavaBeans (EJB) support, Java Message Service (JMS) support
  - spring-orm.jar -- iBATIS SQL Maps support, Apache Object Relational Bridge (OBJ) support, TopLink support, JDO support
  - spring-hibernate.jar -- Hibernate 2.1 support, Hibernate 3.0 and later support
6. Click **Finish** (shown in [Figure 4](#)) to add the new Spring Framework library to the list in the JSF Library References panel of the project Properties. The new Spring Framework library is added to the list on the right, which will be automatically included with your project's .war file.
  7. Click **OK** to apply your changes and close the project Properties dialog box.

## Your task list

To enable the Spring Framework, you need to add a few things to your application:

- A new <listener> in web.xml
- A new <variable-resolver> in faces-config.xml
- A new managed property to your bean; this is how you inject a Spring bean into the existing application
- An applicationContext.xml file that defines the application's beans (including the existing JSF bean)
- A new property for the SignupData bean
- A new Spring-based bean
- Something to show that the new Spring bean is doing something

Lets get to work!

## Add <listener>

1. Add the code from [Listing 6](#) to web.xml before the existing <listener> element as a child of the <web-app> element, not any of the other web.xml tags.

### Listing 6. Enabling the Spring Framework in web.xml

```
<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

The <listener> element adds Spring's ContextLoaderListener to the application; it starts up Spring's root WebApplicationContext, which handles creating and configuring the other parts of Spring that are needed while booting the application.

## Add the Spring variable resolver

2. Save your web.xml file, and right-click the **faces-config.xml** file.
3. Choose **Open With > XML Editor** from the context menu so you can tell your application to use one of the Spring variable resolvers.
4. Add the lines from [Listing 7](#) to faces-config.xml in the <application> element.

### Listing 7. Telling faces-config.xml to use the Spring variable resolver

```
<variable-resolver>org.springframework.web.jsf.DelegatingVariableResolv
er</variable-resolver>
```

The DelegatingVariableResolver will perform lookups for any bean name or bean property references made by the application. If it can't resolve a reference to a Spring bean or property, it delegates the lookup to the standard JSF variable resolver.

## Add a new managed property to your bean

You're going to add a new property to the `signupData` bean declared in this file; the hash property is going to be managed by a Spring bean to demonstrate the technique of injecting Spring into an existing application.

5. Add the lines from [Listing 8](#) to `faces-config.xml` inside of the `<managed-bean>` block that defines the `signupData` bean.

### Listing 8. Adding a managed property to your existing bean

```
<managed-property>
  <property-name>hash</property-name>
  <value>#{genHash.hash}</value>
</managed-property>
```

This creates the new hash-managed property, which maps to the hash property of the new `genHash` bean you're going to define shortly.

## Add the applicationContext.xml file

6. Save your `faces-config.xml` file.
7. Right-click the **WEB-INF** folder in your `devSignup` project, then choose **New > File** from the context menu.
8. Enter `applicationContext.xml` as the file's name, then add the lines from [Listing 9](#) to the new file.

### Listing 9. Defining the beans managed by Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="signupData" class="devSignup.SignupData"
    lazy-init="true"/>
  <bean id="genHash" class="devSignup.HashGenerator"
    lazy-init="true"/>
</beans>
```

These are the beans that Spring will know about, which is why you'll find your JSF bean, `signupData`, here as well. These will be created and managed by Spring's bean factory as necessary. If you'd left the `lazy-init` attribute as the default (`false`), the beans would be created as soon as the application starts. With

lazy-init set to true, the beans are created when they're needed.

## Your Spring-based bean

9. Add the Java code from [Listing 10](#) to the SignupData.java in the devSignup folder; this adds the hash property and its accessors.

Because this is a managed property referring to a property found in another bean, Spring uses it internally to refer to that other bean's property. It won't actually be used by the JSF application, because Spring's `DelegatingVariableResolver` intercepts any references to it.

### Listing 10. Adding the hash property to SignupData.java

```
private String _hash = null;

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

public String getHash() {
    return this._hash;
}

public void setHash( String newHash ) {
    this._hash = newHash;
}
```

You also need to create a new Java class, `HashGenerator`, in the `devSignup` package.

10. Right-click the **src** folder in the `devSignup` project, then choose **New > Other** from the context menu.
11. In the wizard, choose **Class**, then click **Next** to create a new Java class.
12. Set the package to `devSignup` (or click the **Browse** button beside the Package field and choose **devSignup**) and the name to `HashGenerator`.
13. Click **Finish** to create the new class, and edit its source to include the code from [Listing 11](#).

### Listing 11. The new HashGenerator class

```
package devSignup;
```

```
import javax.faces.context.FacesContext;

import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.context.ApplicationContext;
import org.springframework.web.jsf.FacesContextUtils;

public class HashGenerator {
    public void setHash( String newHash ) {
        // Do nothing, this is a read-only bean.
        return;
    }

    public String getHash() {
        ApplicationContext ctx = FacesContextUtils.getWebApplicationContext(
            FacesContext.getCurrentInstance());

        try {
            SignupData myBean = (SignupData)ctx.getBean("signupData" );

            return Integer.toString( myBean.getScreenName().hashCode() );
        } catch( NoSuchBeanDefinitionException ex ) {
            return "no-data-bean";
        } catch( javax.faces.FacesException ex ) {
            return "no-screen-name";
        }
    }
}
```

This `HashGenerator` bean has one read-only property: `hash`. Attempting to set this property has no effect; another option is to log this or maybe use it to seed a more powerful hash-generation method. It also features the first Spring Java code you've seen, in the `getHash()` method.

In the `getHash()` method, you use Spring's `FacesContextUtils.getWebApplicationContext()` method to create an `ApplicationContext` object, which you can then use to look up specific bean instances running in the application. Here you use it to find the `signupData` bean so you can feed its `screenName` property through the Java `String` object's `hashCode()` method. You handle exceptions that might be raised if the bean can't be found or if you've requested an invalid property.

That's all there is to it. Now, when you build and deploy the application, the beans will be running inside of Spring, along with all of the other goodies that Spring provides. A more complex application could take advantage of them quite easily; for example, you could use Spring's support for Hibernate (an object/relational persistence and query service) to store and retrieve user data and add a way for existing users to log in to the hypothetical developer forums.

## Your new Spring-based bean in action

One last thing you should do is add something to the application so you can see this new `HashGenerator` bean in action.

14. Open the `signup-success.jsp` file in Eclipse. As you'll recall, this is the page that gets displayed when someone successfully makes it through the signup page. You'll change the contents of the `<p>` element so it'll display the user's new screen name, and their calculated hash value.
15. Replace the current `<p> . . . </p>` element with the contents of [Listing 12](#).

### Listing 12. Showing the fruits of your labor

```
<p>
Your application was successful, ${signupData.screenName}, welcome to our
forums!
Your hash is ${signupData.hash}.
</p>
```

Now instead of a generic welcome message, they'll be welcomed by name and see the name coming directly from the `signupData` bean and the hash coming from the `HashGenerator` bean referenced by `signupData`'s `hash` property. It looks something like [Figure 5](#).

**Figure 5. Success, now with a more personalized message**



## Developer Forum Signup

Your application was successful, Taffer, welcome to our forums! Your hash is -1040220445.



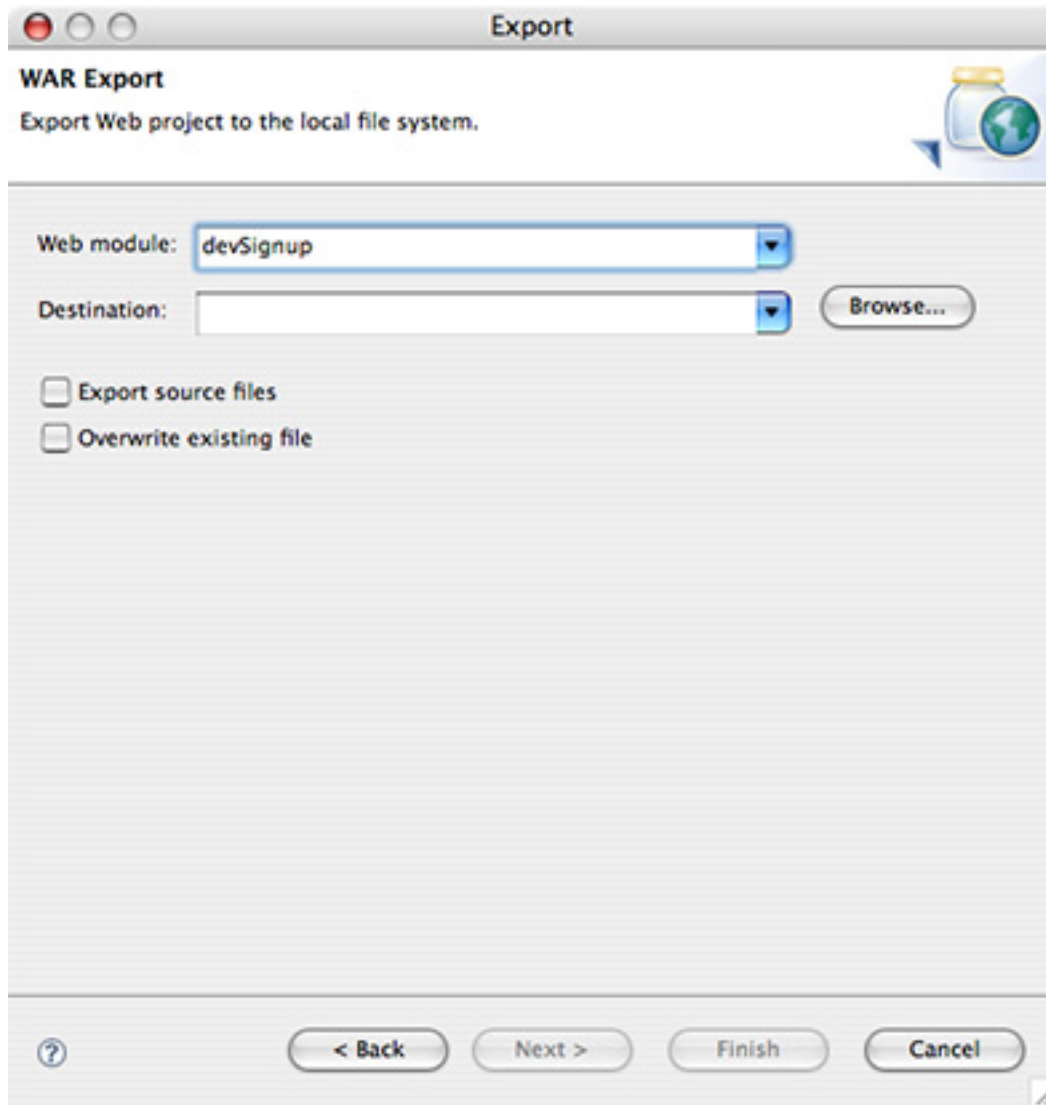
Since you're finished changing the Developer Forum Signup application, you need to deploy it on Geronimo.

## Deploy the new Developer Signup application

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 6](#)).

**Figure 6. 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 will display the Geronimo management console login screen.
7. Log in (remember that the default user name is *server* and the default password is *manager*, and you should change these) to access the administration console.
8. If you've already got devSignup running on your server because you followed along with the previous tutorials, click the **Web App WARs** entry in the Applications section of the list on the left-hand side of the screen. This will give you a list of all installed Web applications, each with a handy **Uninstall** link. Uninstall devSignup before attempting to deploy a new version.
9. Click **Deploy New** from the Applications section in the Console Navigation list to display the Install New Applications screen.
10. Click the **Browse** button next to the Archive field, and browse to your recently-created .war file.
11. Since **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.

That's it, you've successfully extend the Developer Signup Web application using the powerful Spring Framework!

---

## Section 5. Summary

You've taken a very quick look at the Spring Framework and discovered how to add it to an existing JSF application, the Developer Forum Signup example application you started in the [first tutorial](#).

Over the course of this series, you've taken a whirlwind tour of Apache MyFaces and several technologies and frameworks that you can use to extend your JSF Web application, and you've learned how to use the Eclipse IDE's Web Tools Platform to ease the pain of JSF application development. I hope this series has been useful to you!

## Downloads

Description	Name	Size	Download method
Original JSF devsignup app	devSignup-jsf-src.zip	10KB	<a href="#">HTTP</a>
New devsignup app with Spring	devSignup-spring-src.zip	11KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- Read [the official Spring Reference Manual](#), a clean and simple way of learning more about the Spring Framework.
- Get a good introduction to Spring Framework in "[The Spring series, Part 1: Introduction to the Spring framework](#)" (developerWorks, June 2005).
- Get a simple explanation of the Spring Web MVC framework in "[The Spring series, Part 3: Swing into Spring MVC](#)" (developerWorks, September 2005).
- Read "[Why Spring JDBC?](#)," a neat little article on Spring JDBC by Vikram V.
- Read "[Introduction to the Spring Framework](#)" by one of the first and most active members of Spring design team.
- Read "[JSF KickStart: A Simple JavaServer Faces Application](#)," a tutorial that shows you an example of a JSF application developed without any special IDE.
- Read "[A Quick Introduction to JavaServer Faces Plus Apache MyFaces Extensions](#)," a good series of JSF tutorials, including 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).
- See [Sun's JSF tag reference documentation](#) for the h: and f: tags.
- Read the article "[Deploy J2EE applications on Apache Geronimo](#)" (developerWorks, January 2006).
- 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 an [RSS feed for this series](#). (Find out more about [RSS](#).)

### Get products and technologies

- Get [JSR 127](#), the JavaServer Faces specification, straight from Sun.
- 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
- Download Eclipse from [Eclipse.org](#).
- Get Eclipse Web Tools Platform from the [Eclipse WTP Project](#) home page.
- Download the latest version of [Apache Geronimo](#).
- 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 Herboth 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.