

Build Apache Geronimo applications using JavaServer Faces, Part 4: Extend JSF with Apache Trinidad components

Skill Level: Intermediate

[Chris Herborth](#)

Freelance

Freelance Writer

05 Dec 2006

So far in this five-part tutorial series, you've explored Java™ Server Faces (JSF), including deploying a simple JSF application on Apache Geronimo, integrating Apache Tomahawk with the application, and learning how to add Asynchronous JavaScript + XML (Ajax) capabilities to your JSF application using the Sun Ajax4jsf open source framework. In this installment, Part 4, you'll learn how to use Apache Trinidad, the open source version of ADF Faces. Trinidad offers a set of complimentary components that will further enhance the interface of your JSF application.

Section 1. Before you start

This tutorial shows Java programmers how to build highly interactive Java Platform, Enterprise Edition (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 Trinidad, a set of complimentary components that will improve the interface of your JSF application. You'll convert the existing front end of your sample application for the sign-up pages of a developer forum to

use Trinidad components.

About this series

This tutorial is the fourth 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** 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 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.

Prerequisites

This tutorial is for Java developers who have experience accessing back-end systems for data extraction or manipulating. Furthermore, a basic understanding of SQL is required. Ideally, as a developer you'll have experience with a prior version of one or more of the Java Database Connectivity (JDBC) APIs, utilizing interfaces, such as ResultSet and Connection. In addition, you should have a basic understanding of JavaServer Pages (JSP) technology.

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. **Note:** You must use Geronimo 1.2 or later to work with Apache Trinidad.
 - [MyFaces](#), Apache's JSF implementation. Download the core version (without Tomcat) from Apache. We used version 1.1.3 with this tutorial.
 - [Eclipse](#), the extensible open source IDE that supports a wide range of languages and platforms.
 - Apache Trinidad's additional components and input validators for use with any JSF implementation. Based on Oracle's ADF Faces, Trinidad is undergoing Apache incubator development, so you'll have to grab the current development snapshots: Go to the [main snapshot page](#), and select **trinidad-api** and **trinidad-impl** until you find `trinidad-api-incubator-m1-SNAPSHOT.jar` and `trinidad-impl-incubator-m1-SNAPSHOT.jar`.
 - [Eclipse Web Tools Platform \(WTP\)](#), which adds support for XML and JavaScript editing, as well as preliminary JSF support, to Eclipse. Use Eclipse's Update Manager to install the Web Tools Platform from the updater site.
 - [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. A brief review

Before taking a close look at the Trinidad components and how to use some of them in your devSignup example application, let's discuss Trinidad and make sure you have devSignup ready for this tutorial.

Apache Trinidad

Oracle recently donated ADF Faces to the Apache community. The ADF Faces code is being refactored and made more compatible with the Tomahawk components as a new Apache project called Trinidad.

Trinidad provides a set of JSF components that go beyond the ones included in the

specification, while being compatible with any conforming JSF 1.1 implementation (such as MyFaces). These components include support for file uploads, data validators, and converters; an easy way of passing values from one page to another; and a hybrid state-saving strategy that gives you the best of both client- and server-side persistence.

Currently, Trinidad is almost identical to ADF Faces, but it's receiving ongoing community development efforts.

devSignup review

In [Part 1](#) of this series, you learned how to use the Faces Configuration editor in Eclipse to connect JSP processing results with appropriate response files. In [Part 2](#), you reviewed the Apache Tomahawk extensions and added some data validation and an additional UI component to the Developer Forum Signup application. In [Part 3](#), you took Sun's Ajax4jsf tool for a spin and used Ajax techniques to add an interactive temperature conversion utility to the Developer Forum Signup application. Now let's make sure you have devSignup ready to add Trinidad components to the mix.

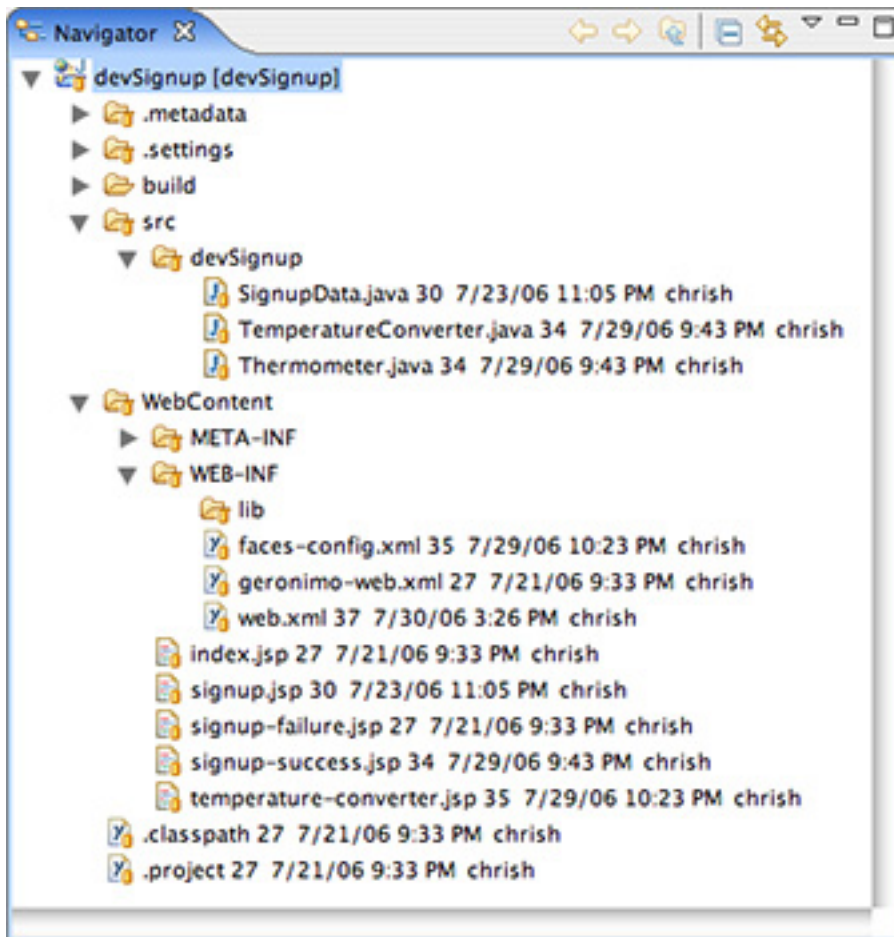
Import the devSignup project

If you didn't read the previous tutorials in this series, you should at least download the devSignup sample project (see the [Download](#) section), because you'll be adding to it later.

[Part 2](#) of this series has detailed instructions for importing the project into Eclipse from the source archive. Download devSignup-src.zip (see the [Download](#) section), and follow the 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 in Part 2, or you won't be able to build the application.

When you're done, your Eclipse Navigator view should look a little like the one in Figure 1.

Figure 1. The devSignup project in Eclipse



If you're new to the Eclipse IDE, you should give yourself a big pat on the back at this point. Importing existing source code projects is one of the things that trips up many new users.

Now you can explore the Trinidad components and start thinking about what they can do for you.

Section 3. Apache Trinidad components

Apache's Trinidad framework provides a lot of useful components, some of which are similar to the components found in Apache's Tomahawk framework. Take a quick look at what each offers the JSF application developer, with an eye towards extending your existing JSF application, the Developer Forum Signup you worked up in the [first tutorial](#) of this series, and that you've been extending ever since.

One important detail: Many of these components include graphical elements (such as images) or JavaScripts that need to be included. They're part of the Trinidad .jar file and are automatically included in the JSP output courtesy of the Trinidad Extension filter. You'll learn about that shortly and about how to install it so you don't have to worry about missing images and JavaScript code required by the Trinidad components.

The UI components

Most of the Trinidad components 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. Here's the rundown:

Table 1. Trinidad components

Component	Description
<code><tr:chooseColor></code>	A component that renders a small color palette the user can use to select a color.
<code><tr:chooseDate></code>	A component used with a <code><tr:selectInputDate></code> component to let the user select a date from a calendar without requiring a second window or calendar page.
<code><tr:column></code>	A column in a table, defining the header, footer, and data for that column.
<code><tr:commandButton></code>	A button that triggers an action.
<code><tr:commandLink></code>	A link that triggers an action.
<code><tr:commandNavigationItem></code>	A menu item that triggers an action.
<code><tr:goButton></code>	A push button that navigates directly to another location instead of triggering an action.
<code><tr:goLink></code>	A component that renders a standard HTML link (<code>...</code>).
<code><tr:commandMenuItem></code>	A menu item that navigates directly to another location instead of triggering an action.
<code><tr:inputFile></code>	A file upload component that supports a label, text, and messages.
<code><tr:inputText></code>	An input text widget that can be single line, multiline, or a password field. It supports a label, text, and messages.
<code><tr:navigationLevel hint="bar" ></code>	A component that creates a bar-shaped navigational menu.
<code><tr:navigationPath></code>	A component that creates a navigational series, indicating the path back to the root page of the

	site (for example Home > Developer Forum Signup > Success, with links on each step returning you to the specified page).
<code><tr:navigationTree></code>	A component that creates a tree-based navigational menu with support for expanding and contracting each level of the tree.
<code><tr:message></code>	A component that displays a message for a component. Most Trinidad input components can automatically show their own messages.
<code><tr:messages></code>	A component that displays page-based messages (as opposed to messages from a specific component). This is usually used near the top of a page to give the user important messages.
<code><tr:objectIcon></code>	A component that renders an icon.
<code><tr:objectImage></code>	A component that renders an image.
<code><tr:objectLegend></code>	A component that renders an icon label.
<code><tr:objectMedia></code>	A component that displays media content, such as audio, video, or images in a player embedded in the Web browser.
<code><tr:objectSeparator></code>	A component that renders a horizontal separator line.
<code><tr:objectSpacer></code>	A component that occupies a fixed amount of space in the page layout. This is better than using "invisible" GIF or PNG images for spacing; it creates a new HTML block element of the specified size instead.
<code><tr:outputFormatted></code>	A component that renders some formatted text, using a subset of HTML's formatting elements.
<code><tr:outputLabel></code>	A component that renders a label for a form component.
<code><tr:outputText></code>	A component that renders some styled text.
<code><tr:page></code>	A component similar to the <code><tr:pagePanel></code> component, but with a model object and a <code><tr:commandMenuItem></code> or <code><tr:goMenuItem></code> stamp for rendering the menu.
<code><tr:panelBorder></code>	A layout component that renders its children in its middle with support for facets that will be rendered in the top, bottom, left, right, start, end, inner-top, inner-bottom, inner-right, inner-start, inner-left, and inner-end areas.
<code><tr:panelBox></code>	A layout component for rendering a boxed group of components, highlighted on the page by a specified background color and an icon.

<code><tr:panelButtonBar></code>	A layout component for rendering a set of buttons.
<code><tr:panelFormLayout></code>	A layout component for rendering input form controls. It lines up their labels and fields vertically, and it supports multiple columns and a footer facet.
<code><tr:panelGroup></code>	A layout component that renders its children as a horizontal or vertical list.
<code><tr:panelHeader></code>	A layout component that renders a label and optional icon at the top of a section.
<code><tr:panelHorizontal></code>	A layout component that renders its children horizontally. The vertical and horizontal alignment can be specified.
<code><tr:panelLabelAndMessage></code>	A layout component that renders a label, child components, a tip, and an associated message.
<code><tr:panelList></code>	A layout component that renders each child as a bulleted list item.
<code><tr:panelPage></code>	A layout component that renders an entire page.
<code><tr:panelPageHeader></code>	A layout component that renders a page header.
<code><tr:panelSideBar></code>	A layout component that renders a side navigation menu.
<code><tr:panelTip></code>	A container for page- or section-level tips.
<code><tr:processChoiceBar></code>	A component that renders a list box with Back or Next buttons on either side for navigation.
<code><tr:processTrain></code>	A component that renders a horizontal list of <i>stations</i> connected by a horizontal line. Each station is rendered as an image and text label. You can use it to represent the user's current stage in a multipage process.
<code><tr:progressIndicator></code>	A component that renders a progress bar to indicate a time-consuming process.
<code><tr:region></code>	A component that renders a previously defined (with <code><tr:componentDef></code>), reusable page region component.
<code><tr:regionDef></code>	A component that defines a region that can be reused on multiple pages.
<code><tr:resetButton></code>	A component that renders a Reset button that will set a form's data back to the default values.
<code><tr:selectBooleanCheckbox></code>	A component that renders a check box with a prompt, text, and messages.
<code><tr:selectBooleanRadio></code>	A component that renders a radio button grouped with the other <code><tr:selectBooleanRadio></code> elements that have the same group attribute. It supports displaying a prompt, text, and

	messages.
<code><tr:selectInputColor></code>	A component that renders an input text field for entering color values, with a button that can display a selectable color palette.
<code><tr:selectInputDate></code>	A component that renders an input text field for entering dates and a button for picking dates from a calendar.
<code><tr:selectInputText></code>	A component that renders an input text field that can also be used to launch an assistance dialog.
<code><tr:selectItem></code>	A component that renders a single item that can be chosen from a list, choice, radio, or shuttle Trinidad control.
<code><tr:selectManyCheckbox></code>	A component that lets the user select one or more items from a series of check boxes.
<code><tr:selectManyListbox></code>	A component that lets the user select one or more items from a list box.
<code><tr:selectManyShuttle></code>	A component that lets the user select one or more items by moving items from one list to another.
<code><tr:selectOneChoice></code>	A component that renders a labeled list box that lets the user select one item from a pop-up list.
<code><tr:selectOneListbox></code>	A component that renders a labeled multiline list box that lets the user select one item.
<code><tr:selectOneRadio></code>	A component that renders a labeled series of radio buttons and lets the user choose one.
<code><tr:selectOrderShuttle></code>	A component similar to <code><tr:selectManyShuttle></code> , except the items in the selected list can be reordered using the first/last and up/down buttons.
<code><tr:selectRangeChoiceBar></code>	A component that lets the user select a range of records and move back and forth using a pop-up list box and buttons.
<code><tr:showDetail></code>	A component that renders a labeled disclosure triangle that lets you show or hide its contents.
<code><tr:showDetailHeader></code>	A component that renders a labeled disclosure triangle and header that lets you show or hide the contents of the header's section.
<code><tr:showDetailItem></code>	An item that can be displayed by one of the <code><tr:showOne*></code> components.
<code><tr:showOneChoice></code>	A component that renders a control that handles a series of items defined by <code><tr:showDetailItem></code> nodes. Displays itself as a label and pop-up list box.
<code><tr:showOneAccordion></code>	A component that renders a collapsing panel control that handles a series of items defined by

	<tr:showDetailItem> nodes.
<tr:showOneRadio>	A component that renders a radio button-based control that handles a series of items defined by <tr:showDetailItem> nodes
<tr:showOneTab>	A component that renders a tab-based control that handles a series of items defined by <tr:showDetailItem> nodes.
<tr:singleStepButtonBar>	A component that renders a control with Back and Next buttons and a label inside describing which step (for example, stage x of y) a user is on.
<tr:table>	A component that renders a table. This component also supports selection (both single and multiple), sorting, record navigation, and detail disclosure.
<tr:group>	An invisible control used to aggregate related elements. The children of the <tr:group> won't be displayed unless the <tr:group>'s parent has support for displaying the grouped children (such as <tr:panelForm>).
<tr:tree>	A component that renders a tree of data, with support for opening and closing each node in the tree.
<tr:treeTable>	A table component for data that also includes a tree.

Obviously, some of these components are really useful, while others might be handy in specific situations. Now take a quick look at the other components in the Trinidad framework.

Other Trinidad components

The Trinidad project also includes several handy components for converting input data to Java types, data validators, and other utility objects.

Table 2. Trinidad components for converting input data

Component	Description
<tr:attribute>	A temporary replacement for the broken <f:attribute> component in JSF 1.1 (which will be fixed in JSF 1.2).
<tr:convertColor>	A component that converts a string to and from a <code>java.awt.Color</code> object.
<tr:convertDateTime>	A component that converts a string to and from a <code>java.util.Date</code> object based on the pattern and style set.

<code><tr:convertNumber></code>	A component that converts a string to and from a <code>java.lang.Number</code> object based on the pattern or type set.
<code><tr:document></code>	A container that creates each of the standard root elements of an HTML page (<code><html></code> , <code><body></code> , and <code><head></code>) in the output.
<code><tr:forEach></code>	A replacement for the JavaServer Pages Standard Tag Library (JSTL) <code><c:forEach></code> tag that works with Trinidad components. Lets you iterate through a <code>java.util.List</code> object or array and, for example, create a <code><tr:selectItem></code> from each.
<code><tr:form></code>	A component that creates an HTML <code><form></code> element in the output.
<code><tr:importScript></code>	A component that imports one or more of the JavaScripts provided by Trinidad. This is useful to make sure that the script is available before using it in a JavaScript handler.
<code><tr:inputHidden></code>	A component that inserts a value that will be submitted with the page, but not displayed to the user.
<code><tr:iterator></code>	A <code><tr:forEach></code> that can be used with JSF <code>DataModel</code> or <code>CollectionModel</code> objects.
<code><tr:panelPartialRoot></code>	A container to enable partial-page rendering for UI component trees or subtrees. There can be multiple <code><tr:panelPartialRoot></code> blocks on a page.
<code><tr:poll></code>	A component that polls the server periodically to update the data model, refresh components, display dialogs, and so on.
<code><tr:resetActionListener></code>	Used inside an action source (such as <code><tr:commandButton></code>) to cause a reset action.
<code><tr:returnActionListener></code>	A component used inside an action source (such as <code><tr:commandLink></code>) to return a value from a dialog or process.
<code><tr:setActionListener></code>	A component used inside an action source (such as <code><tr:commandButton></code>) to set a value before navigation.
<code><tr:subform></code>	A section of a page that can be submitted independently from the rest of the page's contents. The <code><tr:subform></code> 's contents are only processed if a component inside is responsible for submitting the page.
<code><tr:switcher></code>	A component that dynamically decides which facet to render.
<code><tr:validateByteLength></code>	A component that validates the byte-length of strings after encoding.

<code><tr:validateDateTimeRange></code>	A component that validates that the entered date is within the specified range.
<code><tr:validateRegExp></code>	A component that validates the input data using a specified regular expression.
<code><tr:validator></code>	An implementation of the planned JSF 1.2 <code><tr:validator></code> tag. This lets you call back to a bean method to validate the input data.

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

Section 4. Using Trinidad with your Web application

By now you're probably hoping to see some real code to try out a few of these handy components.

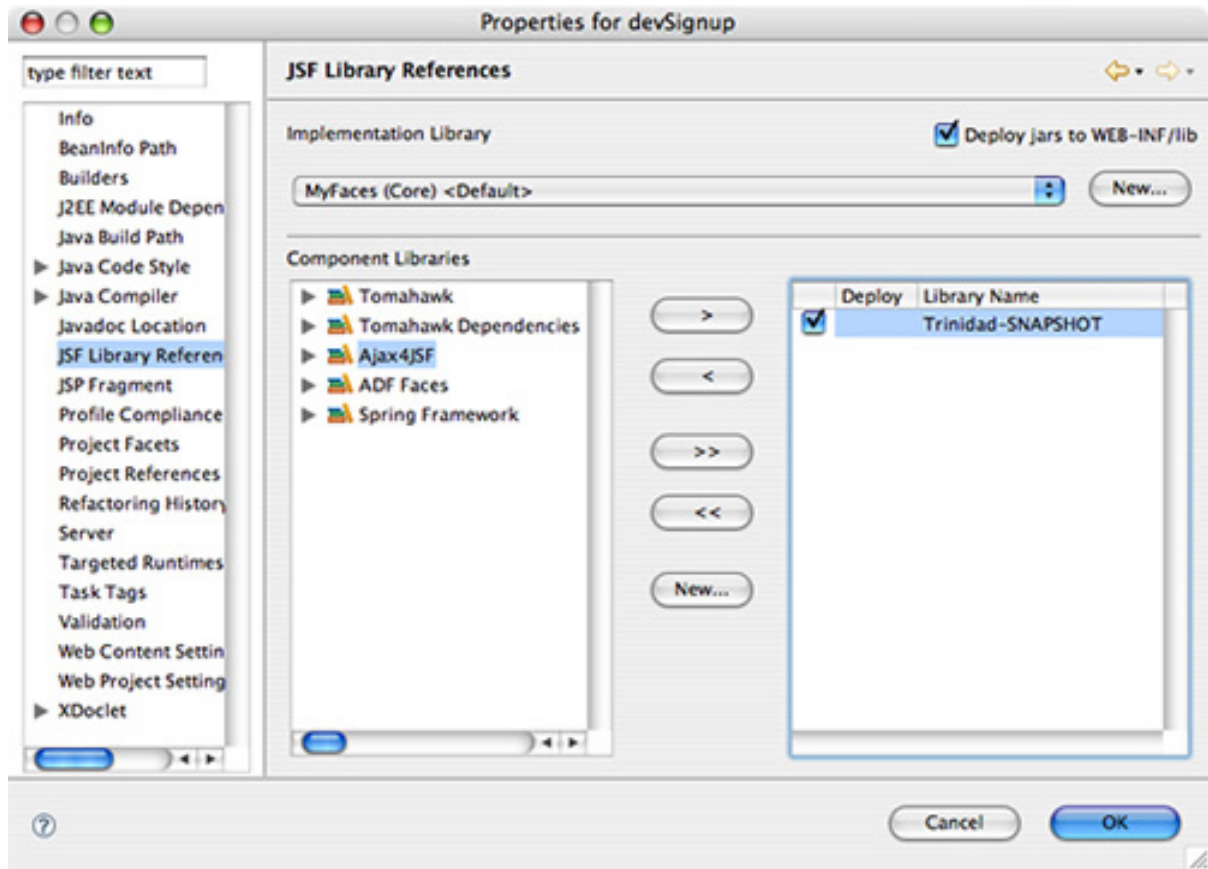
First you'll take a detailed look at what you need to do to use Trinidad with your application, and then you'll extend the Developer Forum Signup application to take advantage of some of the Trinidad components.

Add Trinidad to your project

Adding the Trinidad components to your JSF application is fairly straightforward. You need to add a reference to the Trinidad JAR files, add the extensions filter to your web.xml file, tell JSF to use the Render Kit in the Trinidad library, and include the Trinidad tag library in your JSF pages.

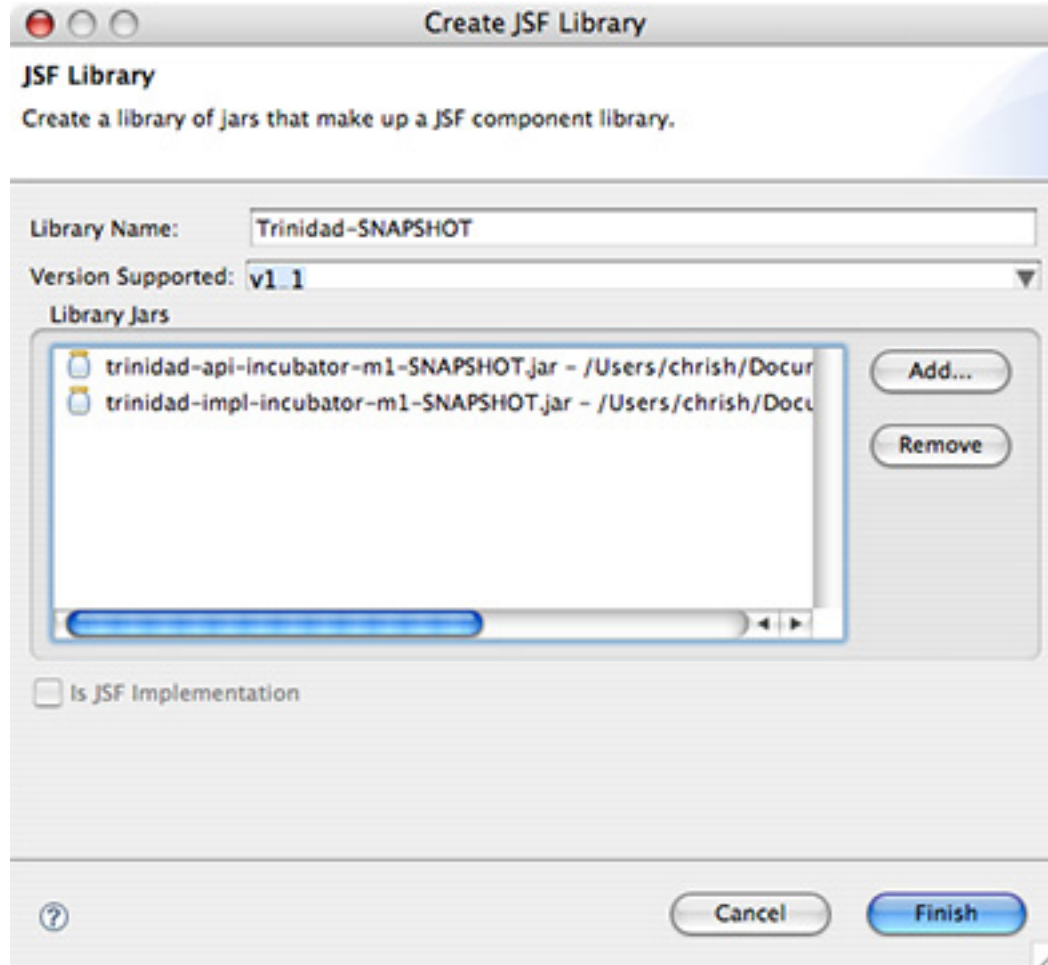
1. Right-click the **devSignup** project in Eclipse's Navigator view, then choose **Properties** from the context menu. Eclipse displays the project properties dialog.
2. Click **JSF Library References** in the list to display the JSF Library References panel (see Figure 2).

Figure 2. Adding Trinidad 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 menu) to display the Create JSF Library dialog box. You use this dialog box 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 using `Trinidad-SNAPSHOT`) in the Library Name field, then choose `v1_1` from the Version Supported drop-down menu, because MyFaces implements JSF 1.1.
5. Click the **Add** button (see Figure 3), then choose the following Trinidad .jar files:
 - `trinidad-api-incubator-m1-SNAPSHOT.jar`
 - `trinidad-impl-incubator-m1-SNAPSHOT.jar`

Figure 3. Making the Trinidad library reference



6. Click **Finish** to add the new Trinidad library to the list in the JSF Library References panel of the project Properties. The new Trinidad library is added to the list on the right, which will be automatically included with your project's .war file.
7. Remove the Ajax4jsf and Tomahawk libraries; we won't be needing them. Ajax4jsf's filters aren't compatible with Trinidad, and most of Tomahawk's functionality is included in Trinidad.
8. Click **OK** to apply your changes and close the project properties dialog box.

To enable the Trinidad extensions, you need to add the extensions filter to your Web application.

1. Double-click the project's **web.xml** file (located in the WebContent/WEB-INF folder, if you've forgotten where it is) 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.

2. Change the `javax.faces.STATE_SAVING_METHOD` value (found in a `<context-param>` block) to `client` if it's set to `server`. Instead of the normal JSF client-side state, you're magically getting the improved Trinidad implementation.
3. Remove the `<filter>` named `MyFacesExtensionsFilter` and any `<filter-mapping>` blocks that refer to `MyFacesExtensionsFilter`; if you're using Trinidad, you don't really need to use the Tomahawk extensions (although you can mix and match them as you wish).
4. 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 Trinidad extensions in web.xml

```
<filter>
  <filter-name>faces</filter-name>
  <filter-class>org.apache.myfaces.trinidadinternal.webapp.FacesFilter</filter-class>
</filter>

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

<servlet>
  <servlet-name>resources</servlet-name>
  <servlet-class>org.apache.myfaces.trinidad.webapp.TrinidadFilter</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>resources</servlet-name>
  <url-pattern>/trinidad/*</url-pattern>
</servlet-mapping>
```

The `<filter>` element adds the Trinidad extension filter class to the list of filters used by your application. These filters all get to process your JSP files before they're rendered to the user's browser.

The `<filter-mapping>` element applies the Trinidad extension filter to any URLs matching the pattern in `<url-pattern>` -- in this case, any reference to a `.faces` page.

The new `<servlet>` and `<servlet-mapping>` elements create a new servlet (resources) that handles requests for resources from any URL matching `/trinidad/*`. This is where Trinidad pulls its JavaScript and other tidbits that need to be sent to the client during processing.

5. Save your web.xml file, and right-click the **faces-config.xml** file.
6. Choose **Open With > XML Editor** from the context menu to tell your application to use the Render Kit in the Trinidad library instead of the default.
7. Add the lines from Listing 2 to faces-config.xml.

Listing 2. Telling faces-config.xml to use the Trinidad Render Kit

```
<application>
<default-render-kit-id>org.apache.myfaces.trinidad.core</default-render
-kit-id>
</application>
```

8. Remove the Temperature Converter Bean and Thermometer Bean's `<managed-bean>` blocks, because you can't use Ajax4jsf with Trinidad.
9. Save your faces-config.xml file.

The last thing you need to do is enable the Trinidad extensions in your JSP files.

1. Add the code from Listing 3 to every JSP page that'll be using Trinidad components. In the devSignup project, that's signup.jsp (the response pages aren't using any JSF code, and we're not going to modify the Ajax-based temperature utility).

Listing 3. Enabling Trinidad in your JSP pages

```
<%@ taglib uri="http://myfaces.apache.org/trinidad" prefix="tr" %>
<%@ taglib uri="http://myfaces.apache.org/trinidad/html" prefix="trh" %>
```

2. You can remove the `taglib` declaration for Tomahawk's components (its URL is `http://myfaces.apache.org/tomahawk`), because we won't be using any Tomahawk components this time.

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

Add to the Developer Forum Signup application

After [Part 3](#), the Developer Forum Signup application is still quite basic. It lets the user enter a screen name, an e-mail address, a password, and the user's birthday (using a fancy Tomahawk calendar pop-up menu). The e-mail address is validated, and you make sure the user entered the password they think they entered by requiring them to enter it the same way twice. Listing 4 shows you the current state of the main signup page, `signup.jsp`.

Listing 4. 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}" 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 verficiation):</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>
```

This is a trivial page, which uses MyFaces components for gathering the user's data and performing the `signupData` bean's `register()` method when the **Sign up** button is clicked. Figure 4 shows you what this might look like in your Web browser.

Figure 4. The current Developer Forum Signup application

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:
 ...

http://localhost:8080/devSignup/signup.faces# 0.992s Adblock

Now, I don't know about you, but to me that's already starting to look a little messy. There are three different JSP tag libraries (the various `f:`, `h:`, and `t:` tags), plus various HTML tags. This isn't a huge application (only around 60 lines), but this could get a little annoying if you need to maintain this for any length of time.

Trinidad includes replacement tags for pretty much everything you've used, and in some cases it implements patterns (such as the `label: entry field` thing

you've used four times) more efficiently. Let's translate this into a pure Trinidad application.

Replace the entire `<h:form>` block with the contents of Listing 5. This is the Trinidad equivalent (without validation). Quite a bit shorter and cleaner, isn't it?

Listing 5. The Trinidad version of the signup form

```
<tr:form>
<tr:panelForm>
  <tr:inputText label="Screen name:" value="#{signupData.screenName}" />
  <tr:inputText label="Email:" value="#{signupData.email}" />
  <tr:inputText label="Password:" value="#{signupData.password}"
secret="true"/>
  <tr:inputText label="Password (for verification):" value=""
secret="true"/>
  <tr:inputDate id="birthday" value="#{signupData.birthday}" />
</tr:panelForm>

<tr:commandButton text="Sign up" action="#{signupData.register}"
immediate="true"/>
</tr:form>
```

The `<tr:inputText>` components create a labeled text-entry field (attached to the given bean property), saving you some extra work compared to the original version. The `<tr:selectInputDate>` and `<tr:commandButton>` are similar to the components you originally used.

Save the file, and you can take a look at what your changes have done to the Developer Signup application.

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.
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 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 management console login screen.
7. Log in (remember, the default user name is `server` and the default password is `manager`, so you should change these) to access the administration console.
8. If you already have `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 side of the screen. This gives 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. 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.

Now when you access the Developer Signup application (`http://localhost:8080/devSignup/signup.faces` on my system; 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 5.

Figure 5. The improved application



Developer Forum Signup

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

Screen name: nobody

Email: nobody@company.com

Password:

Password (for verification): 11/5/06

Sign up

Done 17.691s Adblock

That's it! You've successfully translated the Developer Signup Web application from a MyFaces/Tomahawk hybrid into a pure Trinidad application.

Section 5. Summary

You've explored the Trinidad components, discovered how to add them to your Eclipse Web application project, and used them to rewrite the Developer Forum Signup application you started in the first tutorial.

Stay tuned for Part 5, the final installment in this series, where you'll learn how to integrate your JSF applications with the Spring Framework.

Downloads

Description	Name	Size	Download method
Part 4 source code	devSignup-src.zip	18KB	HTTP

[Information about download methods](#)

Resources

Learn

- Get a good grounding in JSF by reading the [JSF Visual Tutorial](#).
- Check out this [good series of JSF tutorials](#), which includes downloadable JSF applications for testing on your own server.
- Find a wealth of information on [JSR 127](#) straight from Sun Microsystems.
- Check out some Apache goodies, including [Apache Geronimo](#), their Java 2 Platform, Enterprise Edition (J2EE)-certified application server, and their [MyFaces](#) JSF implementation.
- Get the [Trinidad developer's guide](#).
- Join the [Apache Geronimo mailing list](#).
- Find Web Tool Platform 1.5's [list of JSF features](#) (JSF support is 0.5 at the time of this writing).
- See Sun's [JSF tag reference](#) for documentation for the h: and f: tags.
- 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](#).
- Stay current with [developerWorks technical events and webcasts](#).

Get products and technologies

- Peek into the [Apache Trinidad incubator](#) to keep track of pre-release software as it develops into a full-fledged Apache product.
- See the [API changes for Oracle ADF Faces developers migrating to Trinidad](#).
- Download [Eclipse](#).

- Download [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 is a trademark of Sun Microsystems in the United States, other countries, or both.