SAP NetWeaver
How-To Guide

How To... Build UI Custom Components

Applicable Releases:

SAP NetWeaver Composition Environment 7.1

Topic Area:

User Productivity

Development and Composition

Capability:

User Interface Technology

Java

Version 1.0

March 2009

Document History

Document Version	Description
1.00	First official release of this guide

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
`Example text`	File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, F2 or ENTER.

Icons

Icon	Description
	Caution
	Note or Important
	Example
	Recommendation or Tip

Table of Contents

1. Business Scenario

The following guide will cover how to extend JavaServer Faces by creating your own custom UI components. It will introduce a simple “Hello World” custom component to explain the main concepts and the basic issues that you encounter in all custom components.

2. Background Information

JavaServer Faces technology offers a rich set of standard, reusable UI components that enable page authors and application developers to quickly and easily construct UIs for web applications. But often an application requires a component that has additional functionality or requires a completely new component. JavaServer Faces technology allows a component writer to extend the standard components to enhance their functionality or create custom components. For more information about custom UI components you can visit the Java EE 5 tutorial

Before jumping into custom UI component development, it is important to emphasize that creating a UI component is not always necessary. In many cases developers may only need to customize a specific sub-component such as custom converters or validators instead of building a new UI component. Please check the Java EE 5 tutorial to determine whether you need a custom component or renderer.

3. Prerequisites

The following is a list of all you need for developing JSF applications.

- SAP NetWeaver Composition Environment 7.1 AS Java and NWDS.

  Note

  While this tutorial is geared towards to the SAP AS Java (the build/deploy steps of the guide), it wouldn’t be hard to replace the build/deploy portions with similar steps for any other Java EE 5 platform

Knowledge

- You have knowledge of Java Enterprise Edition
- You have acquired experience with JSF applications, for example by working through the JSF tutorials on SDN

4. Step-by-Step Procedure

In the following sections, you will create a simple “Hello World” custom component to understand the basic issues that you encounter while developing any custom component. Before you start building the example, it always helps to summarize the basic steps for creating custom components:

- Create a UIComponent Class
- Register the custom component
- Create UI Component Tag Class
- Create the tag library descriptor (TLD) that defines the custom tag

The following image describes the relationship between the set of sub-components needed to develop a custom component

4.1 Tutorial setup

1. Create a Web Module Development Component named **`hellocomp/web`** as indicated in the Hello World JSF tutorial (Create a Hello World Application using JavaServer Faces [Extern])
2. Create an Enterprise Application Development Component named **`hellocomp/ear`** as indicated in the Hello World JSF tutorial (Create a Hello World Application using JavaServer Faces [Extern])

4.2 Create a UIComponent class

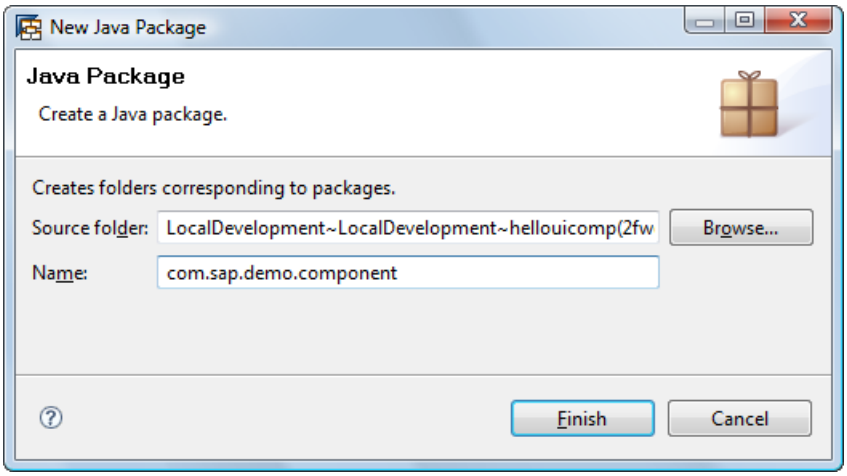
The UI class is a java class that represents the core behavior of the component. It must extend the UIComponent class, which defines over 40 abstract methods, so it is possible to extend an existing class that implements them, such as the UIOutput class.

Recommendation

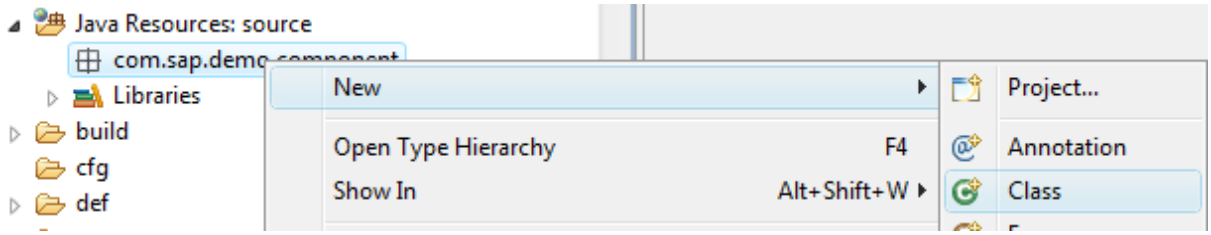
JSF components should be packed to a JAR file with the following directory structure. This recommendation is based on the JSF standard and the packaging of components in the JSF Reference Implementation

```
<EXSAMPLE>.jar
    com
        company name
            ...
                component
                    UI<EXSAMPLE>.class
                Renderkit
                    <EXSAMPLE>Renderer.class
                Taglib
                    <EXSAMPLE>Tag.class
    META-INF
        faces-config.xml
        <EXSAMPLE>.tld
```

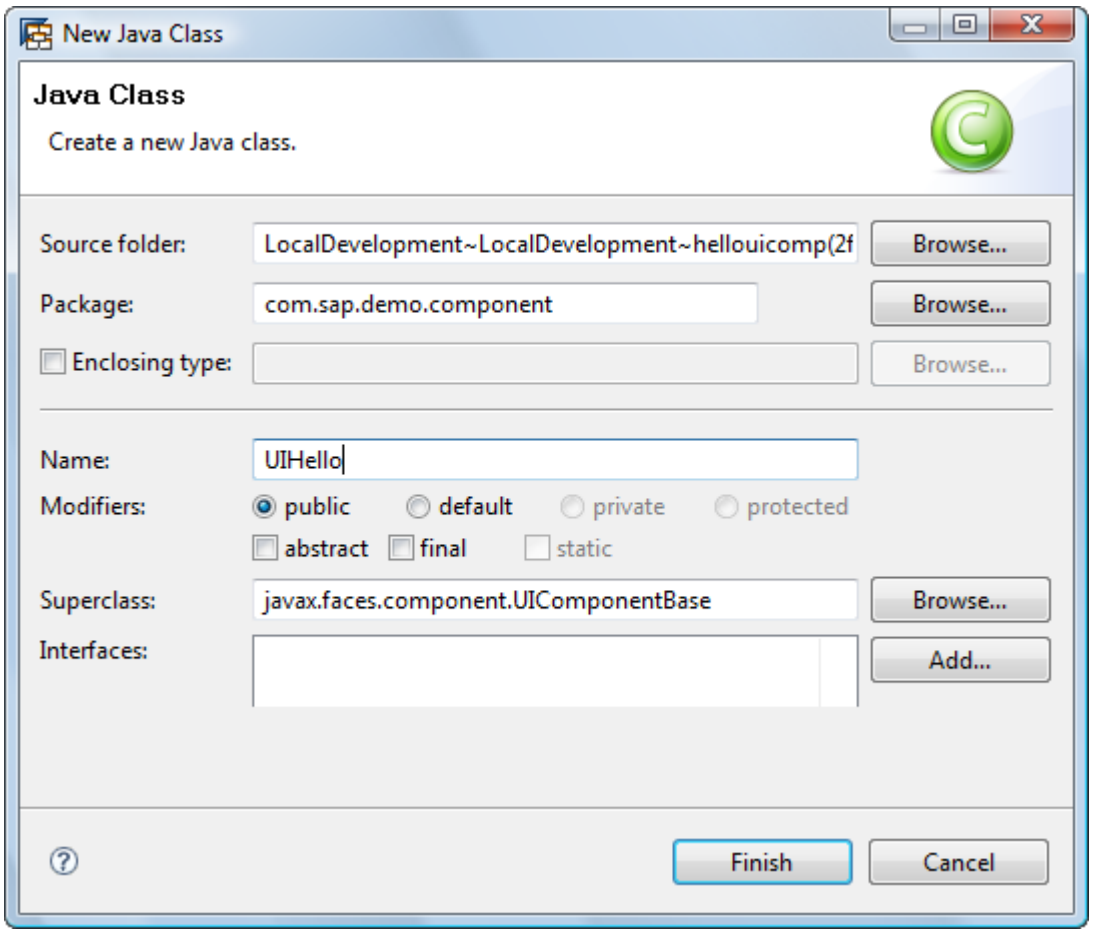
1. Following the recommendation, create the package `com.sap.demo.component`



2. Right-click the **`com.sap.demo.component`** package and select *new* → *class*

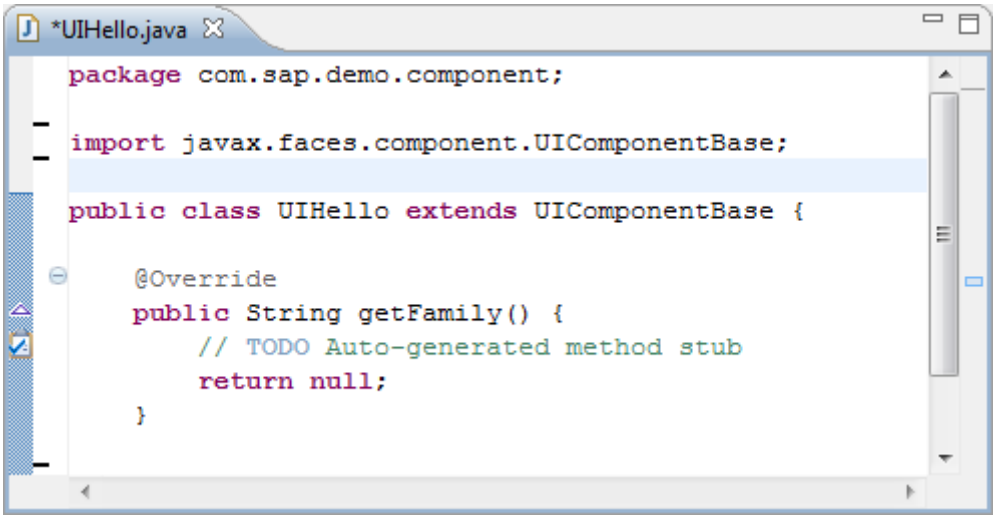


3. Enter **`UIHello`** in the *Name* field and select **`UIComponentBase`** class in the *superclass* field



4. The new class inherits the *getFamily* method of the *UIComponentBase* class. Since in this example you don't need a new family of components, this method can return null.

> **Note**
>
> Notice the *UIHello* class extends the *javax.faces.component.UIComponentBase* class. It is also possible to extend the *UIOutput* class which is basically the same as the *UIComponentBase*, the only difference is the *UIOutput* class has a "value" attribute, that you don't need for the purpose of this example.



5. The *UIHello* class can either render the user interface itself or can delegate the rendering logic to a separate renderer class. In this example it will render itself, so the hello message is embedded in the *UIHello* class in the *encodeBegin* method.

> **Note**
>
> UI Components and/or associated renderer classes can use encode*() methods to display themselves to a client: `encodeBegin()`, `encodeChildren()`and `encodeEnd()`. Since this component doesn't have children, all the encoding can be done in the `encodeBegin` method.

6. Add the encodeBegin method with the following code:

```
public void encodeBegin(FacesContext context) throws IOException {
 ResponseWriter writer = context.getResponseWriter();
 writer.startElement("div", this);
 writer.writeAttribute("style", "color:blue;text-align:center", null);
 writer.writeText("HelloWorld!",null);
 writer.startElement("br", null);
 writer.writeText("Today is: " + new java.util.Date(), null);
 writer.endElement("div");
}
```

> **Important**
>
> In a closer look at the code, you will notice that the *ResponseWriter* class has methods for writing HTML tags. The *startElement* and *endElement* methods produce the element delimiters and the *writeAttribute* method writes an attribute name/value pair with the appropriate escape characters.

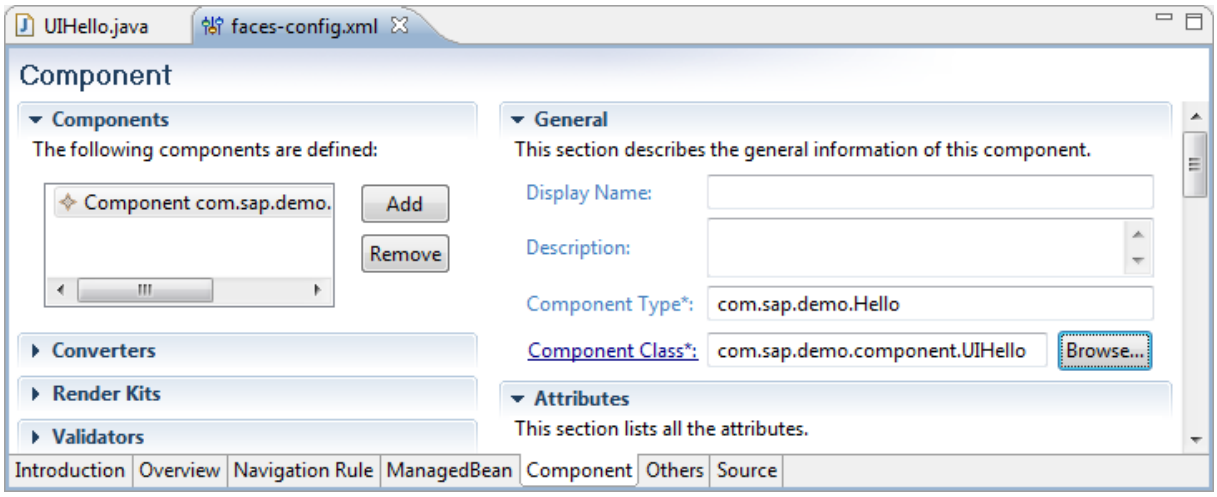
7. Save the changes you have done

4.3 Register the custom component

1. Open the *faces-config.xml* file
2. Select the *Component* tab

3. In the *Components* section, click the *Add* button and enter **`com.sap.demo.Hello`** in the *Component type* field and **`com.sap.demo.component.UIHello`** in the *Component Class* field

> **Note**
>
> The *component-type* is the JSF name of the *UIHello* custom component and it will be used in the UI Component Tag Class
> The *component-class* is the actual class path address of the *UIHello* custom component



4. The following code will be inserted automatically in the *source* tab

```
<component>
        <component-type>com.sap.demo.Hello</component-type>
        <component-class>com.sap.demo.component.UIHello</component-class>
</component>
```

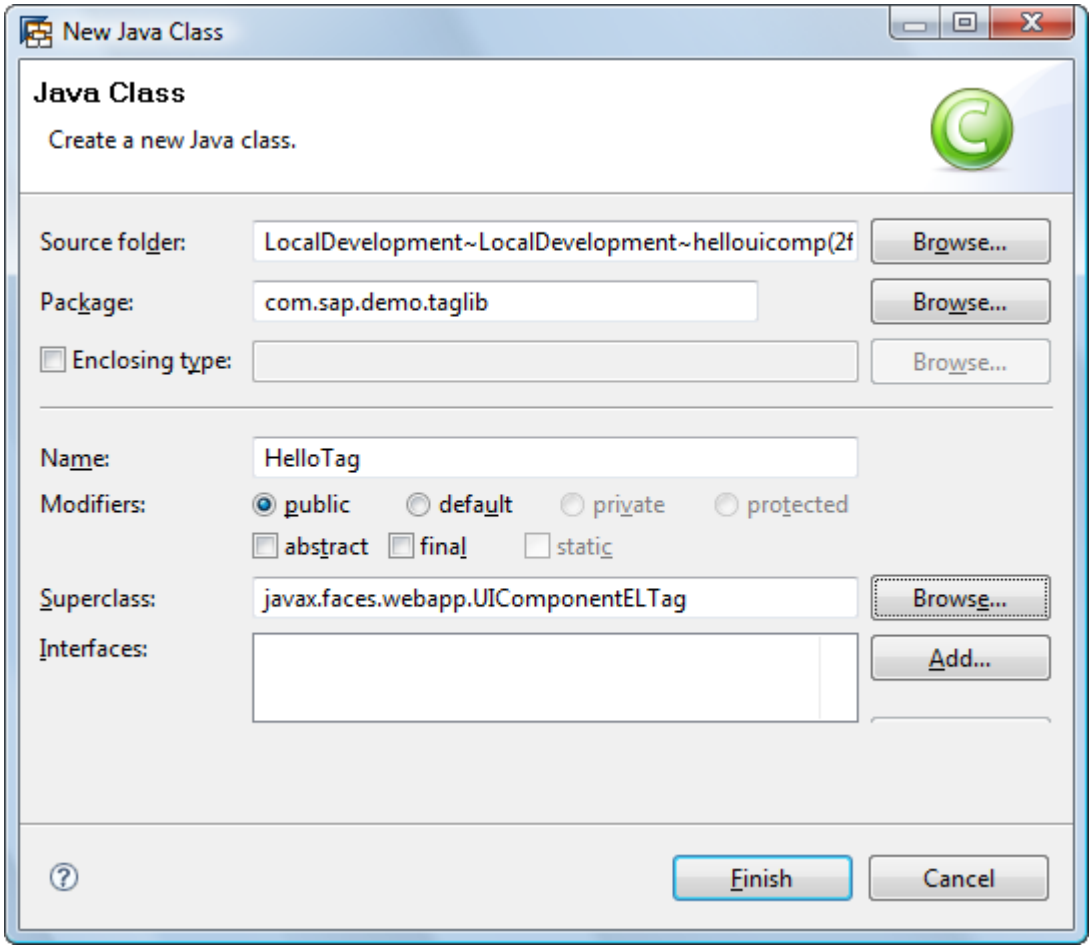
5. Save the changes

4.4 Create UI Component Tag Class

This is a JSP tag handler class that allows the UI Component to be used in a JSP deployment environment. It can also associate a separate renderer class with a UIComponent class

1. Create a Java class with the following configuration

> **Note**
>
> Notice
> The package name is com.sap.demo.taglib
> The JSP taghandler class extends the javax.faces.webapp.UIComponentELTag class



2. The new class inherits two methods from the UIComponentELTag class: *getComponentType and getRendererType.*

> **Note**
>
> The *getComponentType* method associates this tag handler with registered UI Component
> The *getRendererType* method associates the renderer

3. In this example there is no renderer class, so the *getRendererType* method can return null. Implement the methods as shown in the following code

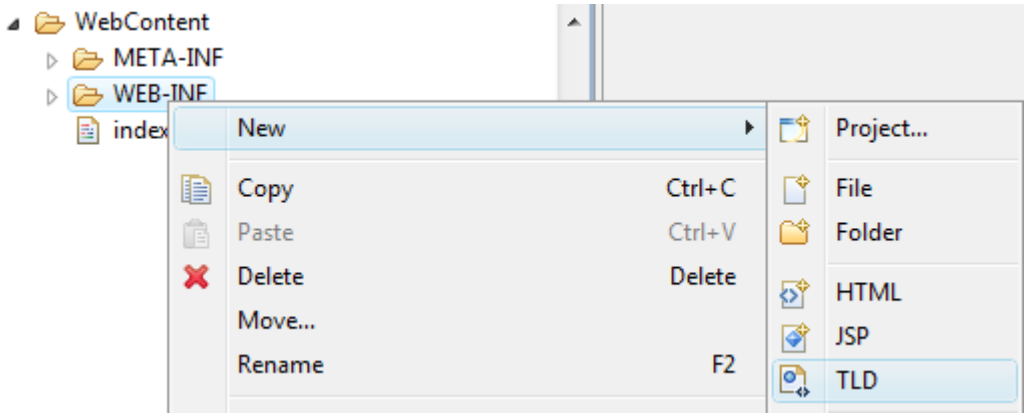
```
public class HelloTag extends UIComponentELTag {
  @Override
  public String getComponentType() {
     return "com.sap.demo.Hello";
  }
  @Override
  public String getRendererType() {
     return null;
  }
}
```

4. Save the changes

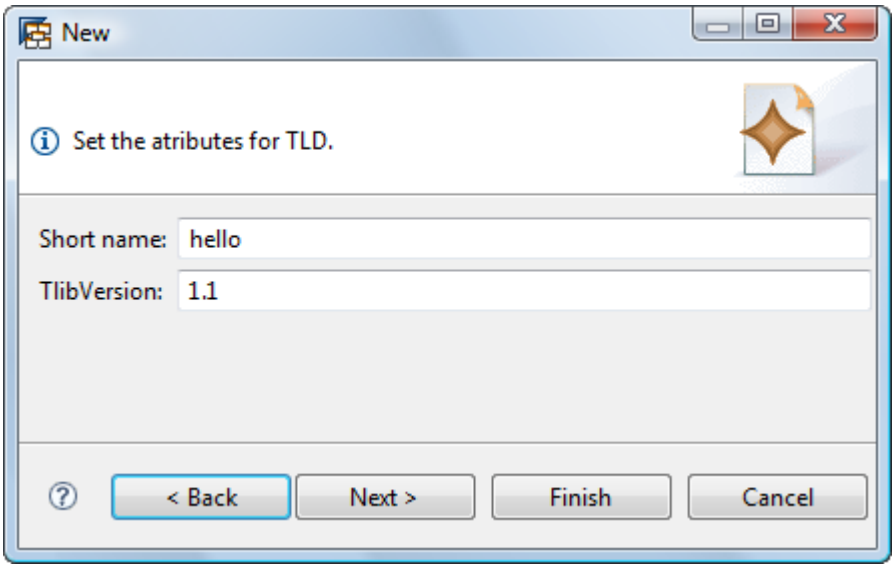
4.5 Create the tag library descriptor (TLD)

The tag library descriptor specifies the class name for the tag handler and the permitted attributes of the tag.

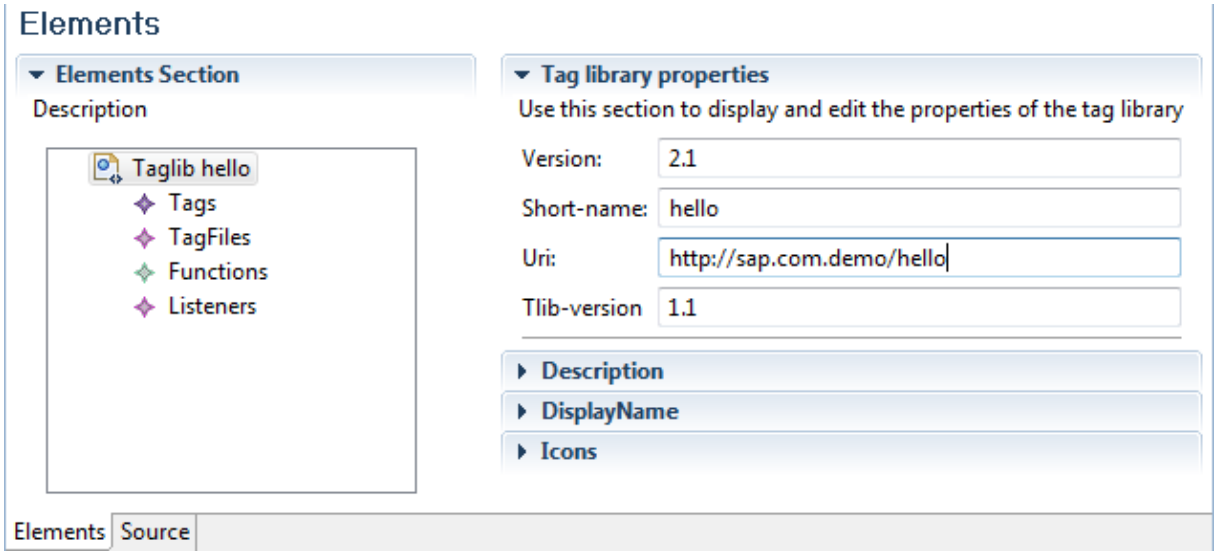
5. From the context menu of the *WEB-INF* folder in the Web Module project select *New → TLD*



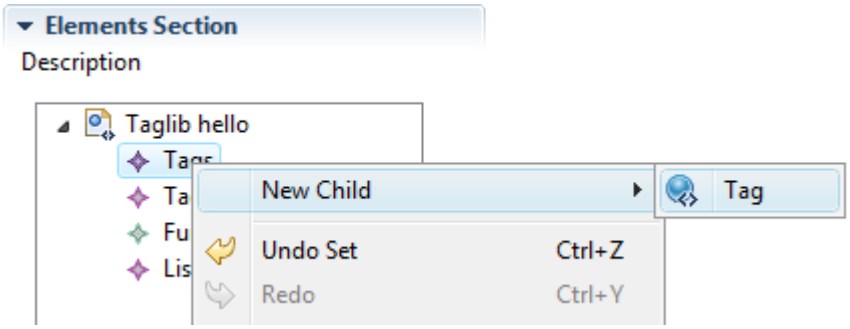
6. Enter **`hello.tld`** in the *Name* field and click the *Next* button.
7. Enter **`hello`** in the *Short name* field and **`1.1`** in the *TlibVersion* field and click the *Finish* button



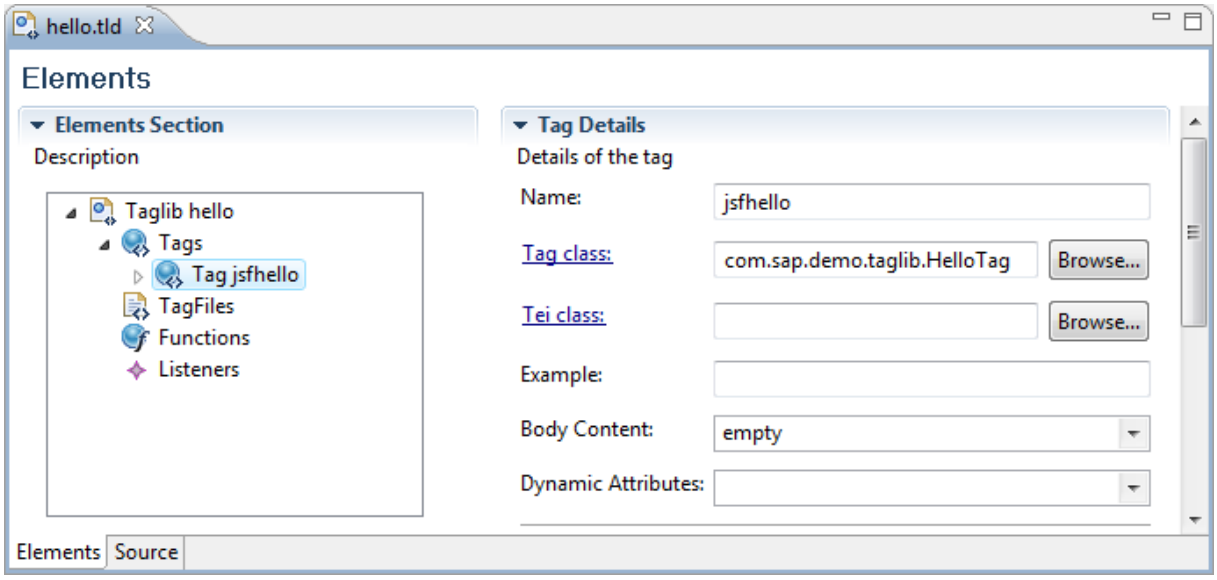
8. The NWDS will open the *tld* file.
9. In the *Elements section*, select *Taglib hello* element and add **`http://sap.com.demo/hello`** in the *Uri* field



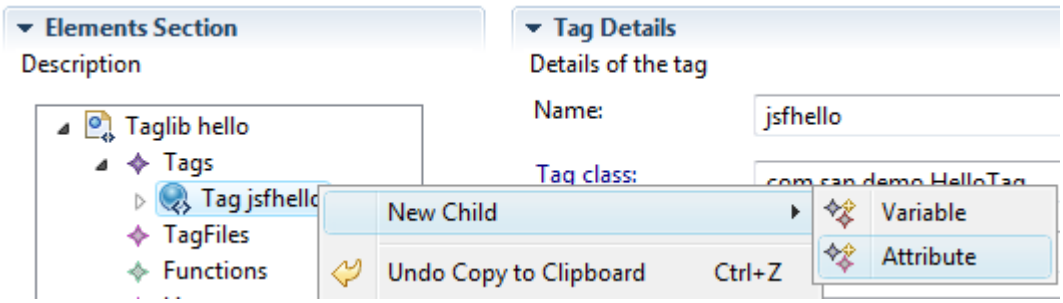
10. In the context menu of the *Tags* element, select *New child* → *Tag*



11. The Tag Details editor will open. Enter **`jsfhello`** in the *Name* field, **`com.sap.demo.taglib.HelloTag`** in the *Tag class* field and **`empty`** in the *Body Content* field

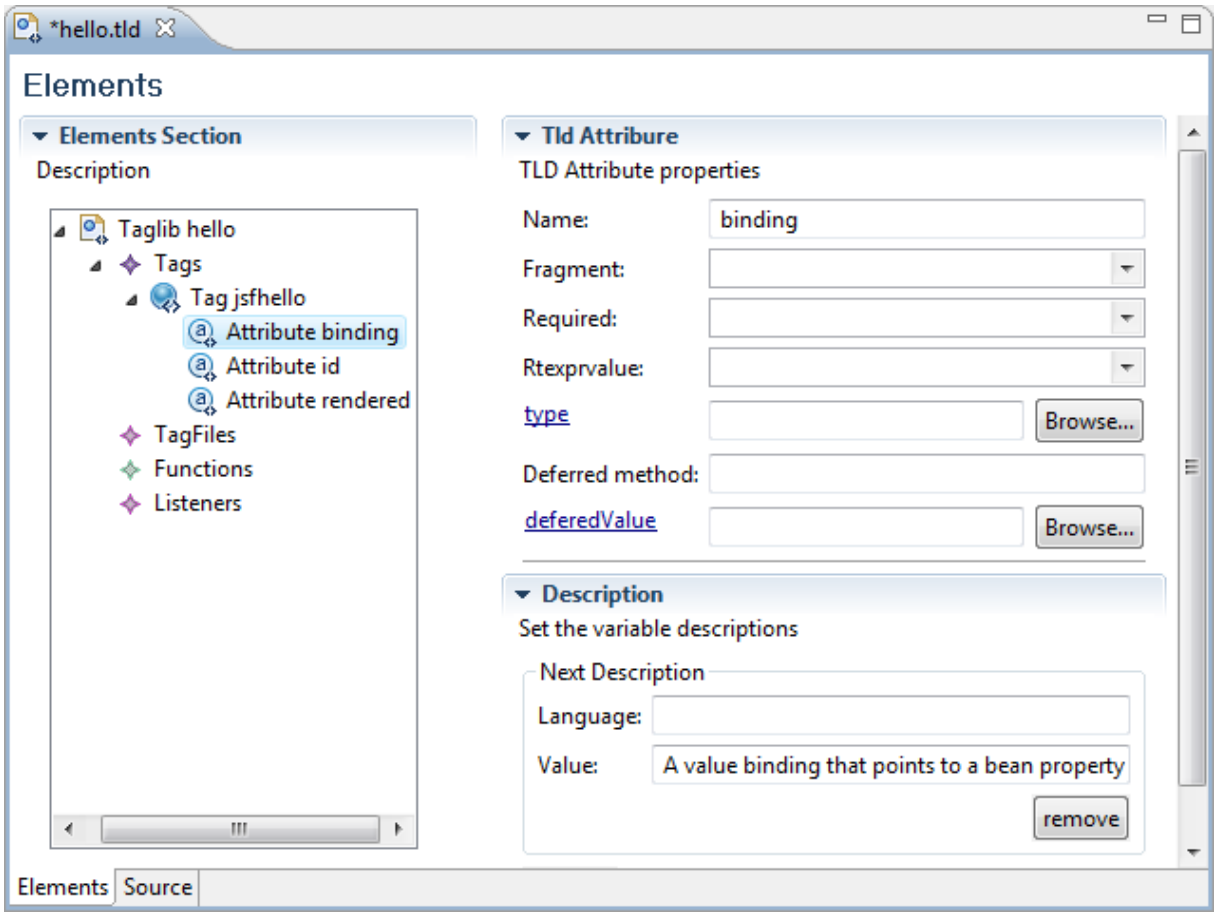


12. In the context menu of the *Tag jsfhello* element, select *New Child* → *Attribute*



13. Create three attributes with the following values

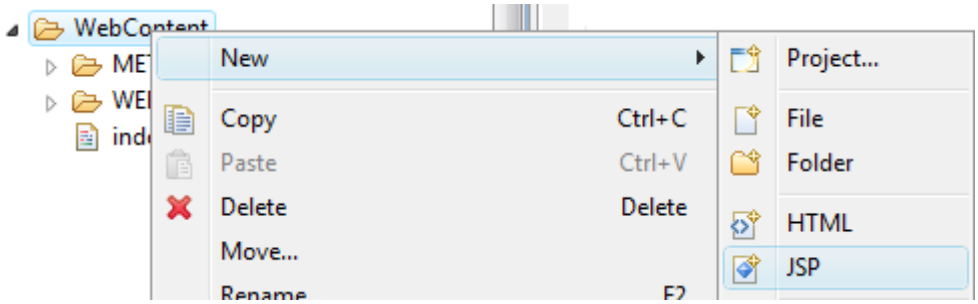
Name	Description
binding	A value binding that points to a bean property
id	The client id of this component
rendered	Is this component rendered?



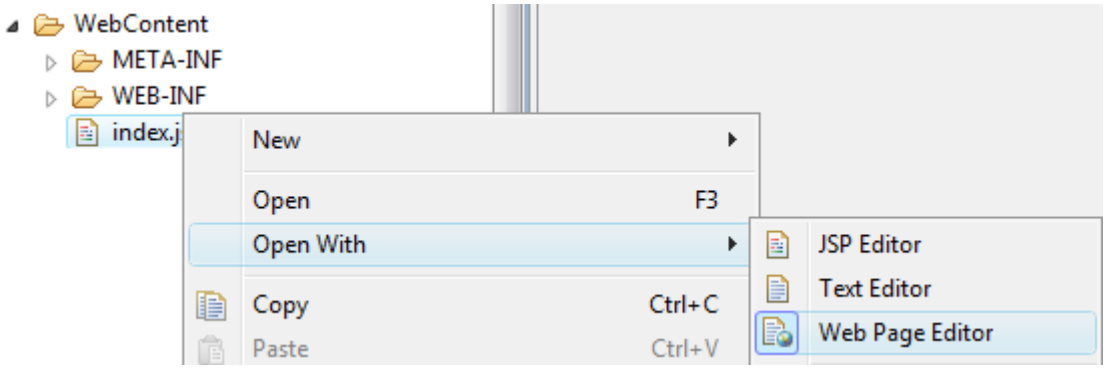
14. Save the changes

4.6 Test your component

1. Create a JSP page called index.jsp in the *WebContent* folder



2. Open the *index.jsp* with the Web Page Editor



3. Place an extra taglib directive specifying a custom tag library (tld) which contains the custom tag

```
<%@taglib uri="http://sap.com.demo/hello" prefix="cf"%>
```

4. Place the *jsfhello* tag into the body of the JSP

```
<body>
<f:view>
    <h:form>
        <p>The HelloWorld UI Component:</p>
            <cf:jsfhello/>
    </h:form>
</f:view>
</body>
```

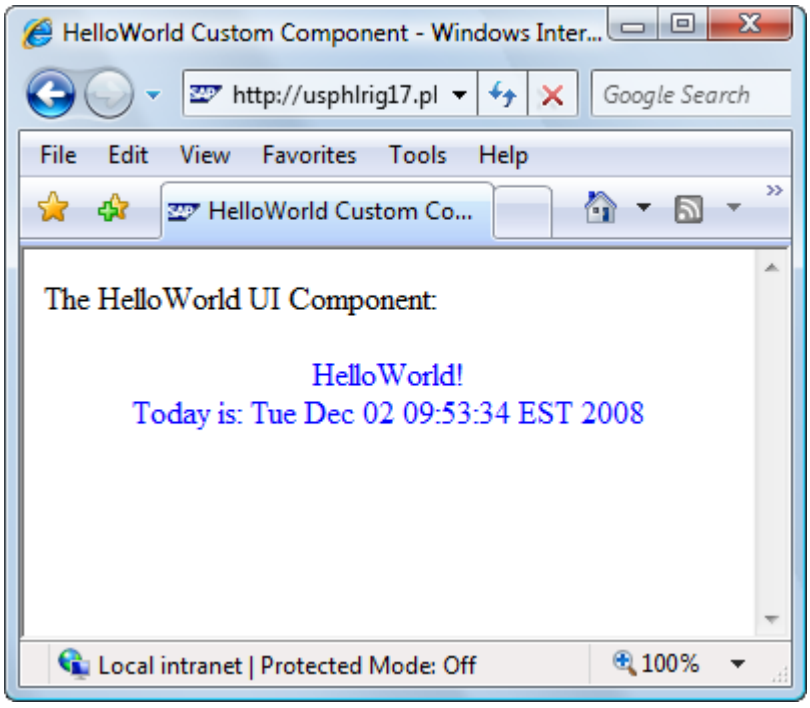
5. Save the changes

4.7 Build, Deploy and Run your application

6. Create the application.xml deployment descriptor, sets the WAR file to “demo.sap.com~hellocomp~web.war” and the context root to “hellocomp” as indicated in the Hello World JSF tutorial (Create a Hello World Application using JavaServer Faces [Extern])
7. Save changes.
8. Build and deploy the application.
9. Run the application using the following simplified URL:

http://<servername>:<httpport>/hellocomp/faces/index.jsp

10. Result:



www.sdn.sap.com/irj/sdn/howtoguides