



# 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

© Copyright 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, Informix, i5/OS, POWER, POWER5, OpenPower and PowerPC are trademarks or registered trademarks of IBM Corporation.

Adobe, the Adobe logo, Acrobat, PostScript, and Reader are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice.

These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

SAP NetWeaver "How-to" Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

#### Disclaimer

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

## Document History

Document Version	Description
1.00	First official release of this guide

## Typographic Conventions

Type Style	Description
<i>Example Text</i>	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
<b>Example text</b>	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.
<b>Example text</b>	User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.
<b>&lt;Example text&gt;</b>	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**..... 1
- 2. **Background Information**..... 1
- 3. **Prerequisites** ..... 1
- 4. **Step-by-Step Procedure**..... 2
  - 4.1 Tutorial setup ..... 3
  - 4.2 Create a UIComponent class ..... 3
  - 4.3 Register the custom component..... 5
  - 4.4 Create UI Component Tag Class ..... 6
  - 4.5 Create the tag library descriptor (TLD)..... 8
  - 4.6 Test your component..... 11
  - 4.7 Build, Deploy and Run your application ..... 11

## 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 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

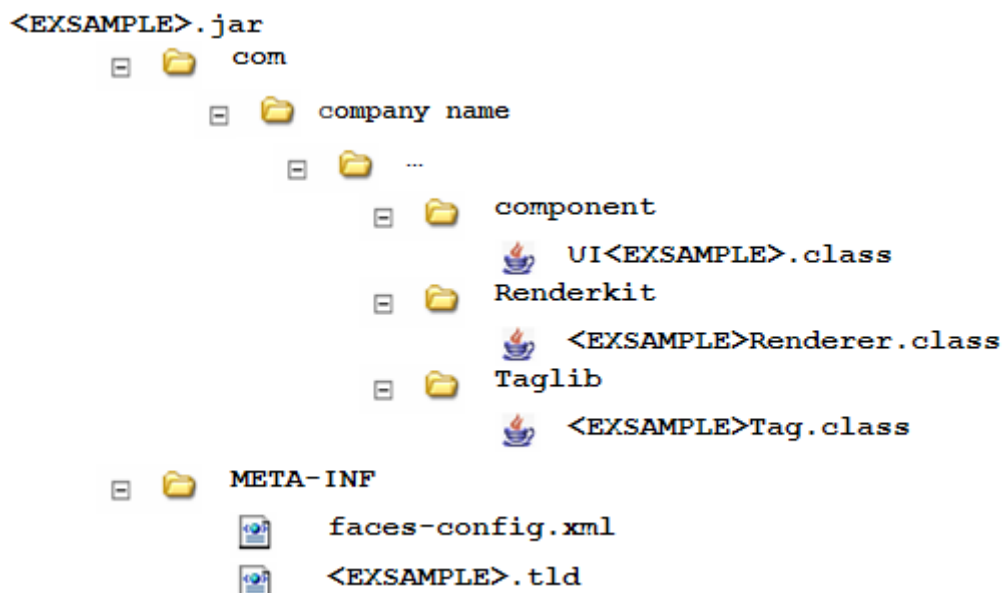
1. Create a Web Module Development Component named `hellocomp/web` as indicated in the Hello World JSF tutorial ([Create a Hello World Application using JavaServer Faces \[Extern\]](#))
2. Create an Enterprise Application Development Component named `hellocomp/ear` as indicated in the Hello World JSF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

## 4.2 Create a UIComponent class

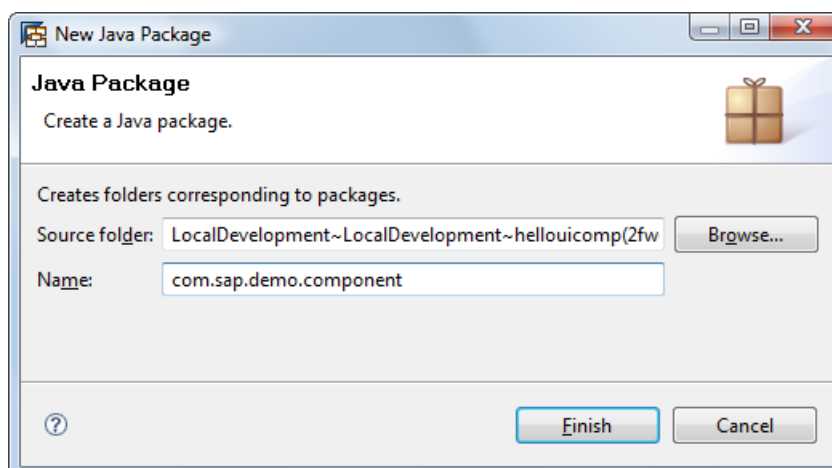
The UI class is a java class that represents the core behavior of the component. It must extend the `UIComponent` class, which defines over 40 abstract methods, so it is possible to extend an existing class that implements them, such as the `UIOutput` class.

### 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

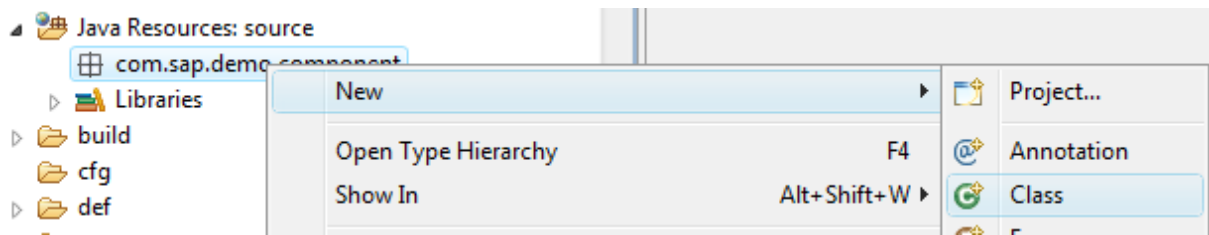


1. Following the recommendation, create the package `com.sap.demo.component`

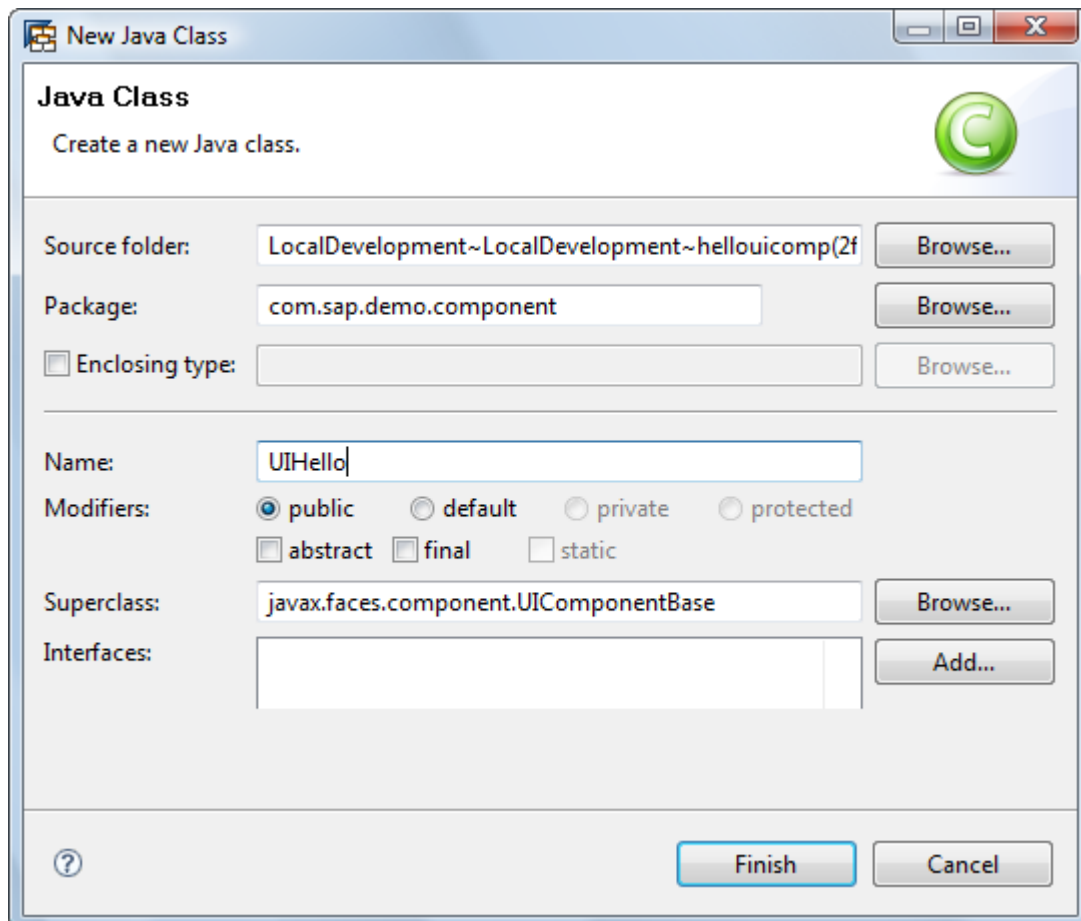




- Right-click the `com.sap.demo.component` package and select *new* → *class*



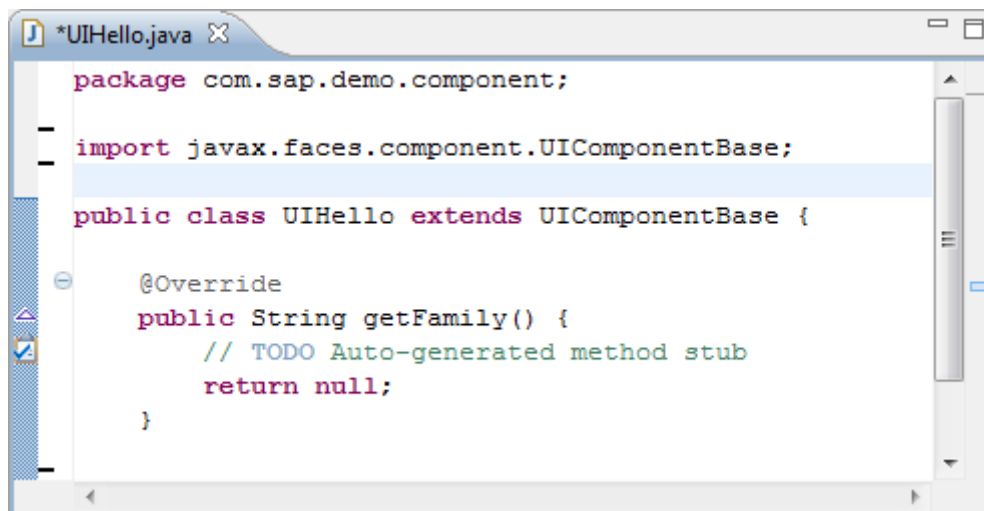
- Enter `UIHello` in the *Name* field and select `UIComponentBase` class in the *superclass* field



- 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.



```
package com.sap.demo.component;

import javax.faces.component.UIComponentBase;

public class UIHello extends UIComponentBase {

    @Override
    public String getFamily() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

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

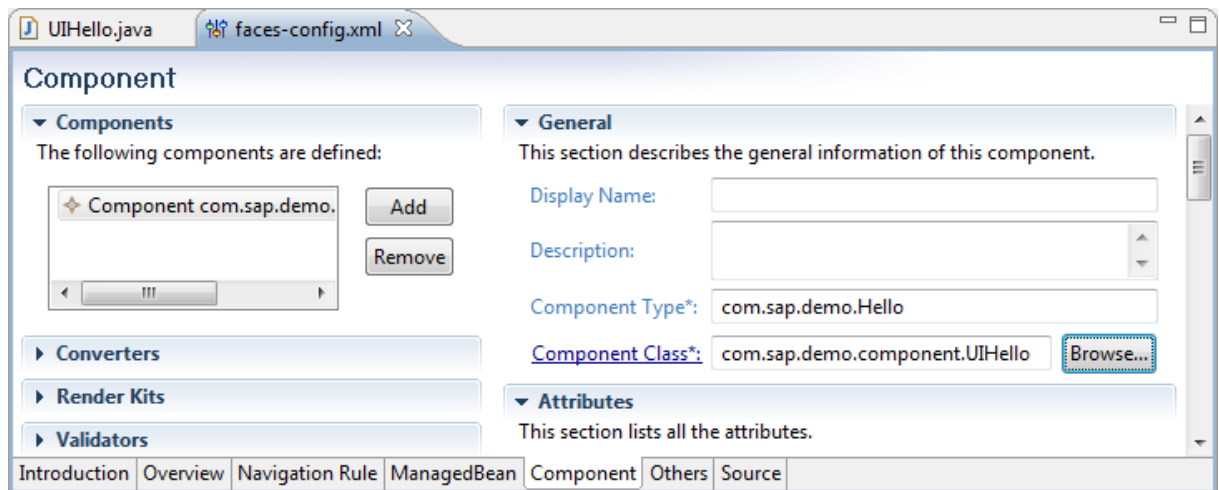
1. Open the *faces-config.xml* file
2. Select the *Component* tab

- 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



- 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>
```

- 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

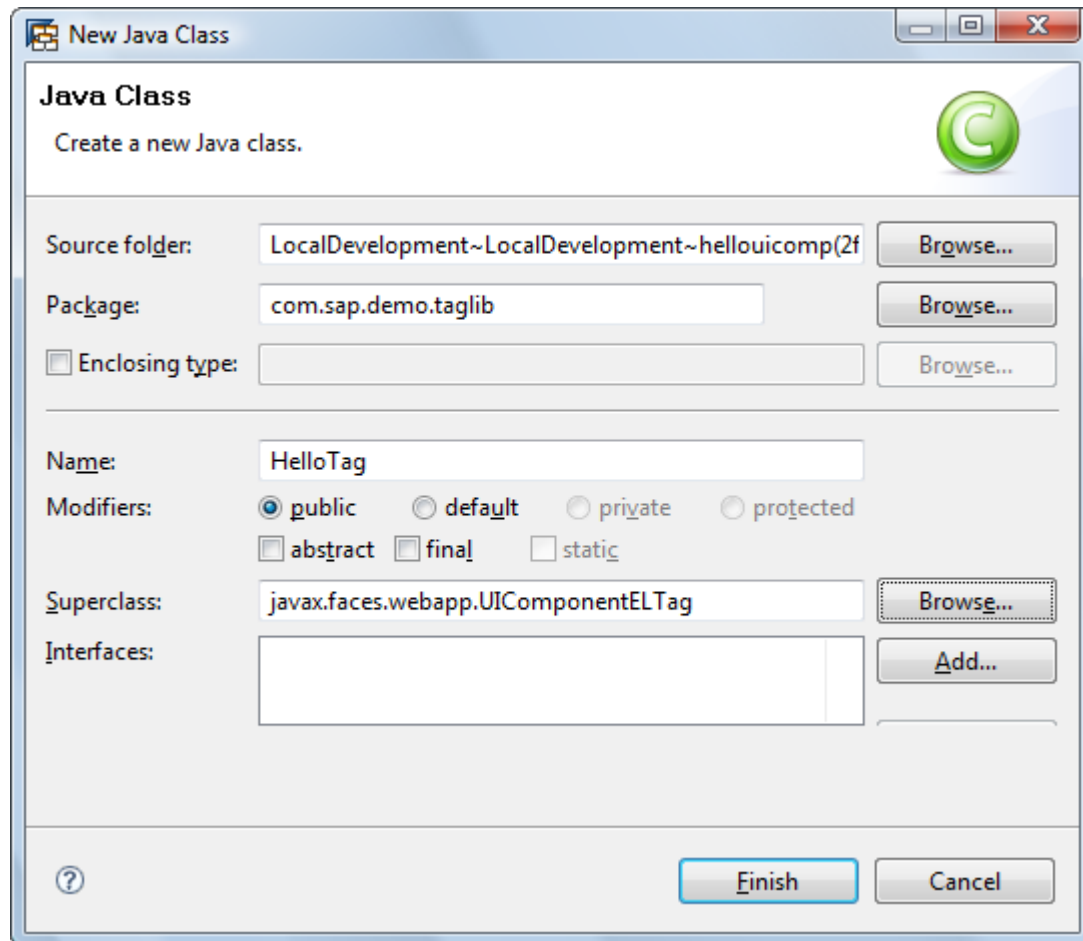
- Create a Java class with the following configuration

 Note

Notice

The package name is `com.sap.demo.taglib`

The JSP taghandler class extends the `javax.faces.webapp.UIComponentELTag` class



- 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

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

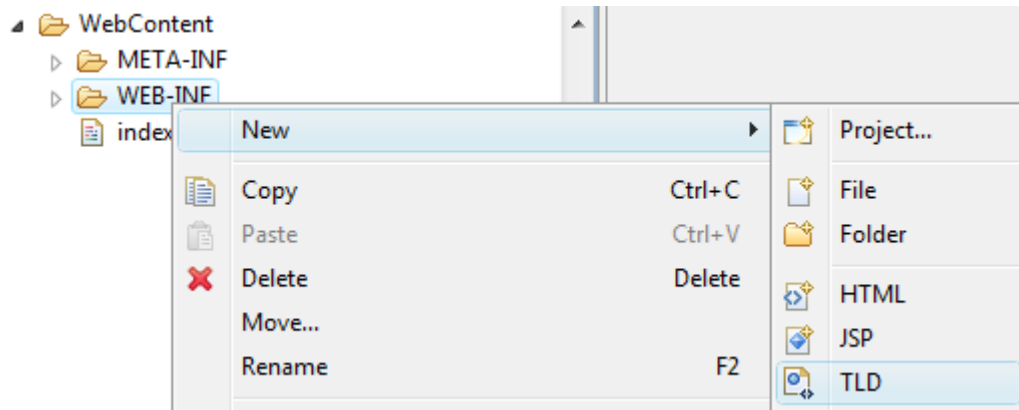
```
public class HelloTag extends UIComponentELTag {
    @Override
    public String getComponentType() {
        return "com.sap.demo.Hello";
    }
    @Override
    public String getRendererType() {
        return null;
    }
}
```

4. Save the changes

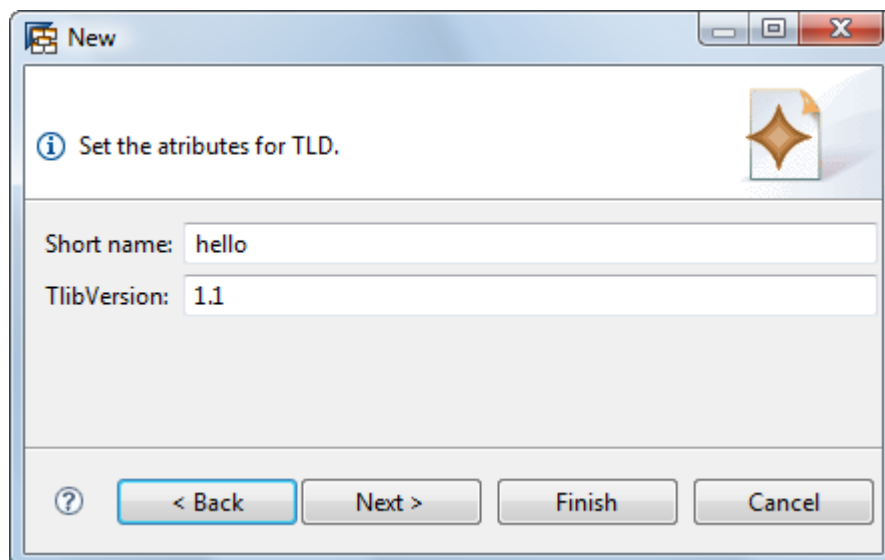
## 4.5 Create the tag library descriptor (TLD)

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

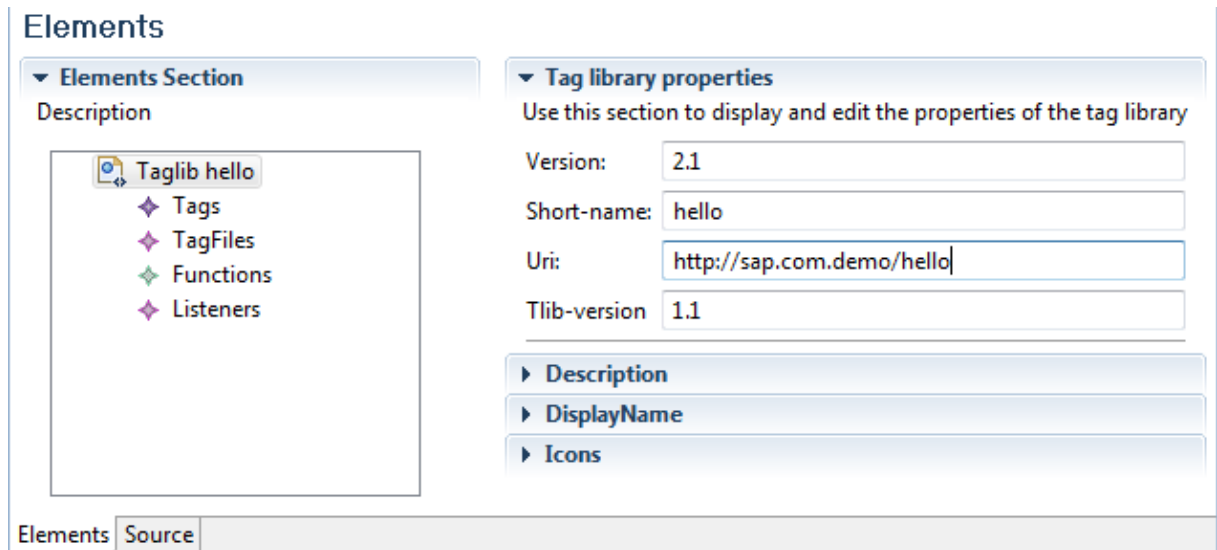
5. From the context menu of the *WEB-INF* folder in the Web Module project select *New* → *TLD*



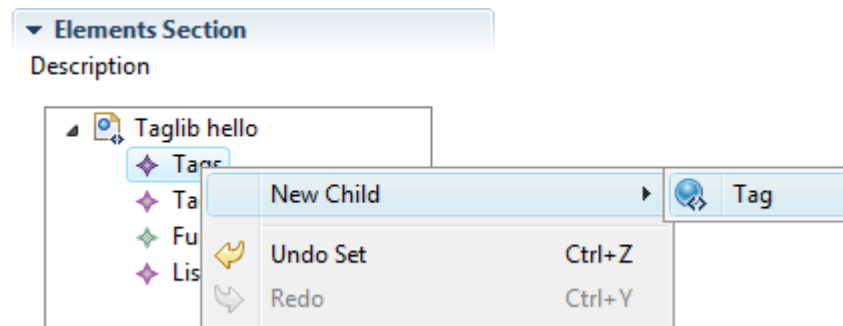
6. Enter `hello.tld` in the *Name* field and click the *Next* button.
7. Enter `hello` in the *Short name* field and `1.1` in the *TlibVersion* field and click the *Finish* button



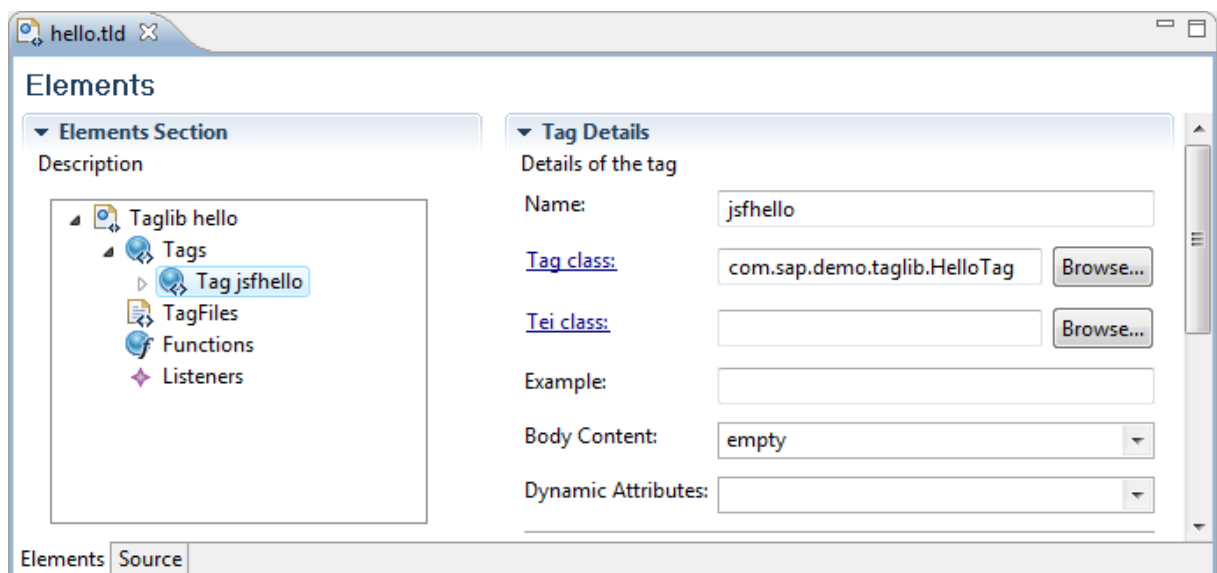
8. The NWDS will open the *tld* file.
9. In the *Elements* section, select *Taglib hello* element and add `http://sap.com.demo/hello` in the *Uri* field



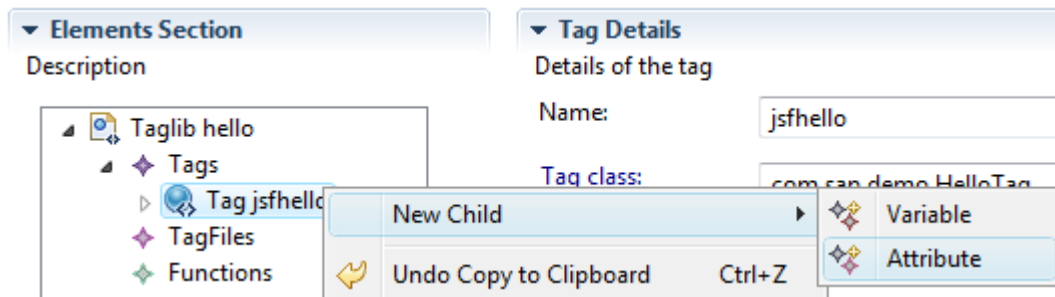
- In the context menu of the *Tags* element, select *New child* → *Tag*



- 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

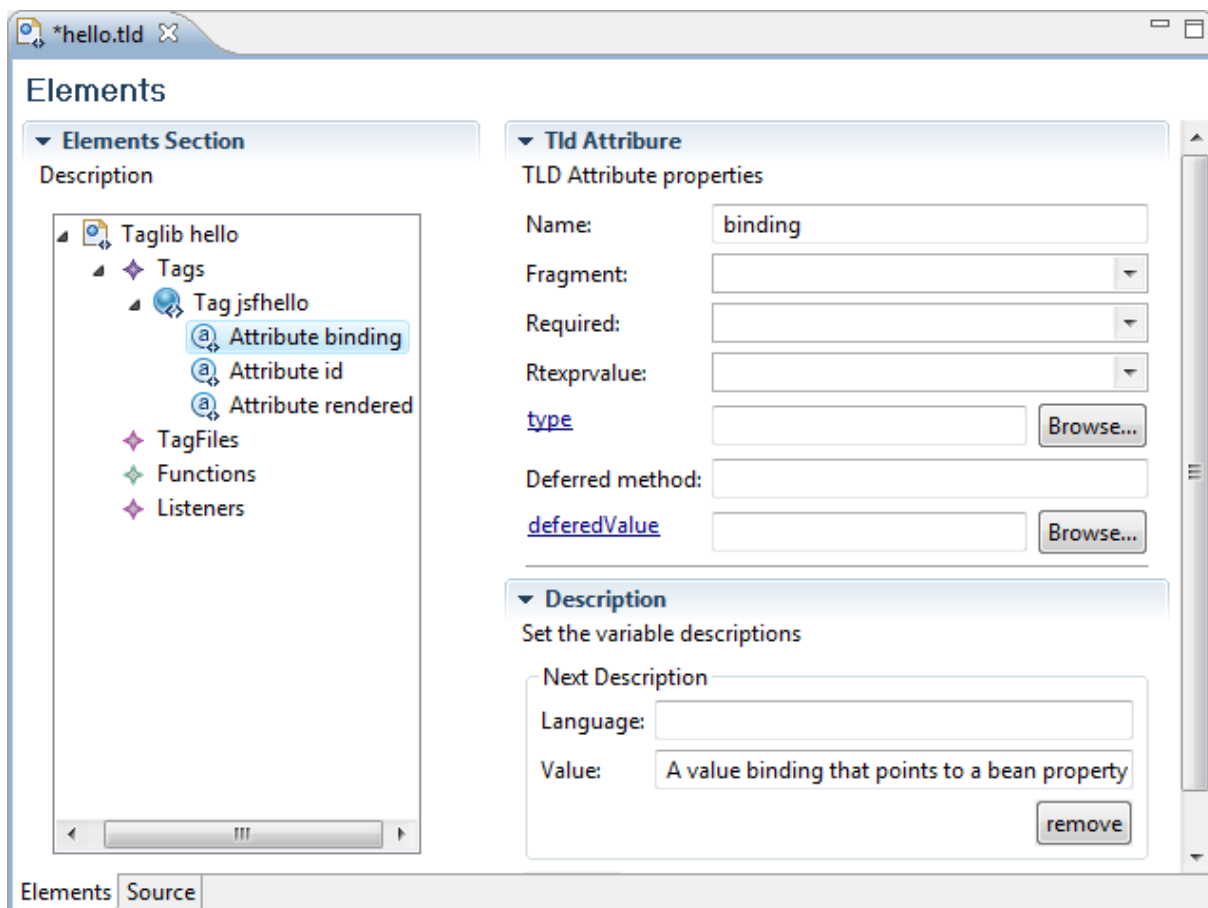


12. In the context menu of the *Tag jsfhello* element, select *New Child* → *Attribute*



13. Create three attributes with the following values

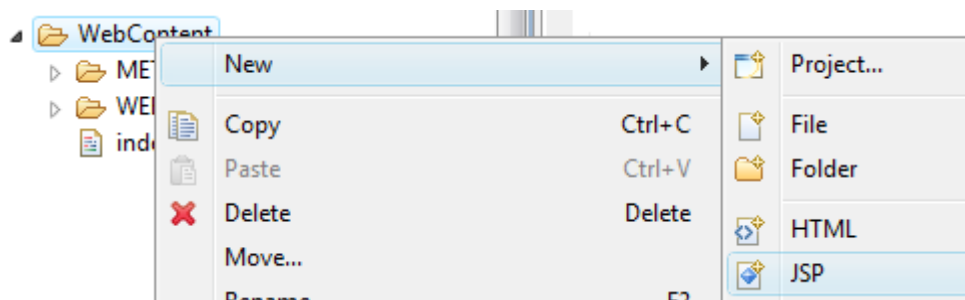
Name	Description
binding	A value binding that points to a bean property
id	The client id of this component
rendered	Is this component rendered?



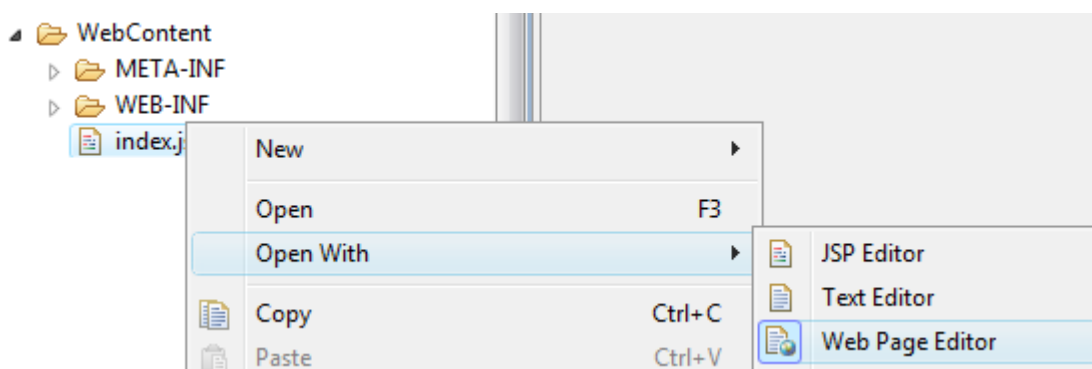
14. Save the changes

## 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

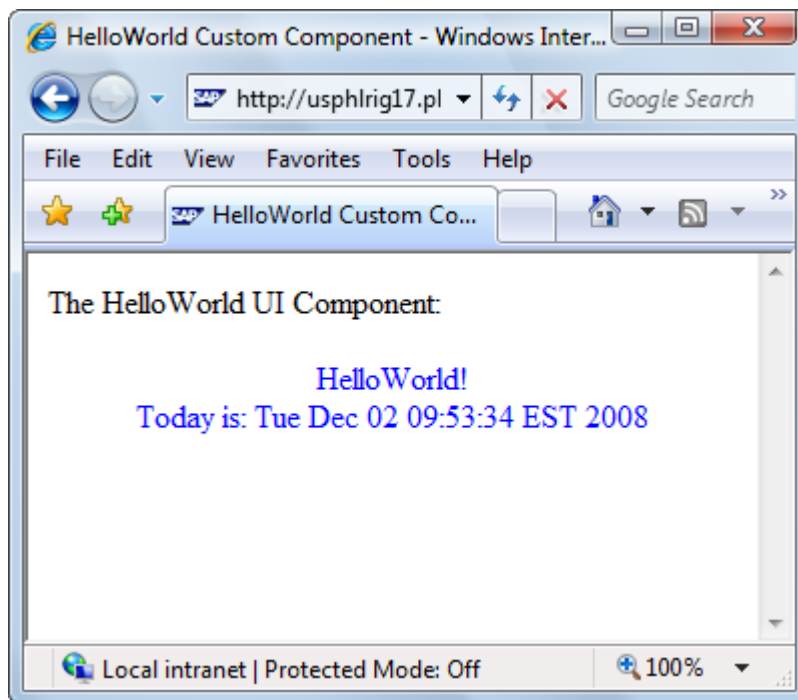
## 4.7 Build, Deploy and Run your application

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](http://www.sdn.sap.com/irj/sdn/howtoguides)