Creating a custom JSF 2.0 taglib

Jonas Freiknecht, 2012 Karlsruhe

www.jofre.de

Abstract

To create custom JSF components is pretty easy but it definitely lacks documentation. We begin with a simple introduction to the two kinds of components:

1. **Composite Components** are assembled from existing JSF components and consist to a huge extend of XHTML code.
2. **Classic Components** are written in Java and supported by the strong JSF annotation framework. Their advantage is that – once they are compiled – their source code is hidden.

Focusing on classic components, this tutorial will show how to write and package them into a JAR so that they can be used in any other JSF project.

1. Project setup

We need a Dynamic Web Project to test the component and a basic Java Project to create the taglib. In illustration 1 *VisualFaces* is the Java Project and *VisualFacesImpl* is our test project.

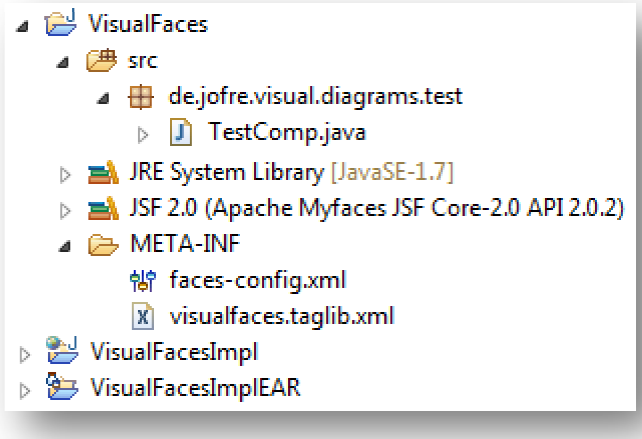

Illustration 1: Project setup

I already added the three necessary components:

1. In the source folder I created a component that is discussed in the following chapter.
2. The *META-INF* directory that contains two essential files:
    a. The *faces-config.xml* shows that the JAR has to be regarded as being interesting for the JSF runtime.

b. The *visualfaces.taglib.xml* exports all components that are offered by the taglib[1].

Please create the *META-INF* directory as well as the file *faces-config.xml* and copy & paste the code from listing 1.

```
<?xml version="1.0" encoding="UTF-8"?>

<faces-config
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
    version="2.0">
</faces-config>
```

Listing 1: faces-config.xml

2. Writing a component in Java

Create a package i.e. **de.jofre.visual.diagrams.test**, add a class called **TestComp** and paste the following code.

```
@FacesComponent("TestComp")
public class TestComp extends UIComponentBase {

	@Override
	public String getFamily() {
		return "jofre.visual.diagrams";
	}


	@Override
	public void encodeBegin(FacesContext context) throws IOException {
		if (context == null) {
			throw new NullPointerException();
		}
		String strMsg = (String)getAttributes().get("msg");
		ResponseWriter writer = context.getResponseWriter();
		writer.startElement("p", this);
		writer.writeText("This is a test: "+strMsg, null);
	}

	@Override
	public void encodeEnd(FacesContext context) throws IOException {

		if (context == null) {
			throw new NullPointerException();
		}
		ResponseWriter writer = context.getResponseWriter();
		writer.endElement("p");
	}
}
```

Listing 2: The component TestComp

[1] To make the JSF runtime recognize a taglib as such the file name has to end with ".taglib.xml". The beginning does not matter.

To be able to use the annotation *FacesContext* you will have to add a JSF Library. Therefore, simply enable the JSF facet for your Dynamic Web Project *VisualFacesImpl*. Eclipse will automatically ask you to download the necessary libraries (if you have not done it so far). The let Eclipse fix your project imports (by clicking on the lamp next to the *FacesContext* error) and add the downloaded library. According to the *faces-confix.xml* in listing 1, you should choose a JSF 2.0 implementation.

The source code is simple. We create a component called **TestComp** which extends the very basic component **UIComponentBase**. You now have to implement the **getFamily** method which returns a string containing the name of the component family. In this case it is called **jofre.visual.diagrams** because the component is supposed to be a part of a visualization framework.

The two methods **encodeBegin** and **encodeEnd** are called while the JSF component is rendered. In this case, we request the content of an attribute called **msg** and write it to the string **strMsg**. The **responseWriter** is responsible for writing the finished HTML-code. He opens a tag **p** and adds the content of the **msg** attribute including a dumb prefix to see if the message is drawn even if **strMsg** is empty. **encodeEnd** simply closes the opened **p** tag and we are done here.

3. Registering the component in the visualfaces.taglib.xml

Now, we create the file **visualfaces.taglib.xml** in the **META-INF** directory.

```
<facelet-taglib version="2.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facelettaglibrary_2_0.xsd">

  <namespace>http://www.jofre.de/visualfaces</namespace>
  <composite-library-name>visualfaces</composite-library-name>

      <tag>
            <tag-name>testcomp</tag-name>
            <component>
                  <component-type>TestComp</component-type>
            </component>
      </tag>
</facelet-taglib>
```

Listing 3: visualfaces.taglib.xml

The final content is shown in listing 3. The namespace is important because it is used later on to reference the taglib.

Everyting embraced by the **tag** tag defines a new component. In this case its name is **testcomp**. This name is used to reference it at the end in the XHTML pages. The **component-type** tag tells which java class should be taken to construct the component. The name **TestComp** is the name that is used in the *@FacesContext* annotation.

4. Exporting and using the component

Now that the component is written it has to be packaged as a JAR. Right click the project and select **Export… → Jar file** and save it locally (i.e. as **visualfaces.jar**). Then drag & drop the JAR to the **WEB-INF/lib** folder in the Dynamic Web Project. The *taglib* should be identified as such and the components inside can now be used.

Create an **index.xhtml** in the **WebContent** folder and add a namespace reference and a tag to test the component as shown in listing 4.

```
<!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"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:j="http://www.jofre.de/visualfaces"
      xmlns:f="http://java.sun.com/jsf/core">

<body>
      <j:testcomp msg="asd" />
</body>
</html>
```

Listing 4: Adding the component to index.xhtml

The namespace http://www.jofre.de/visualfaces as defined in the **visualfaces.taglib.xml** is now used and mapped to the prefix **j**. Via this reference you can now add the **testcomp** and furthermore add the attribute **msg**.

The highly interesting output can be seen in illustration 2.

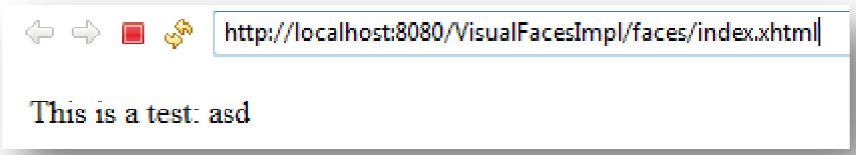

Illustration 2: The rendered output of the testcomp

If you are uncertain about the structure of the Dynamic Web Project you can have a look at illustration 3. **visualfaces.jar** is the export of the library we just created.

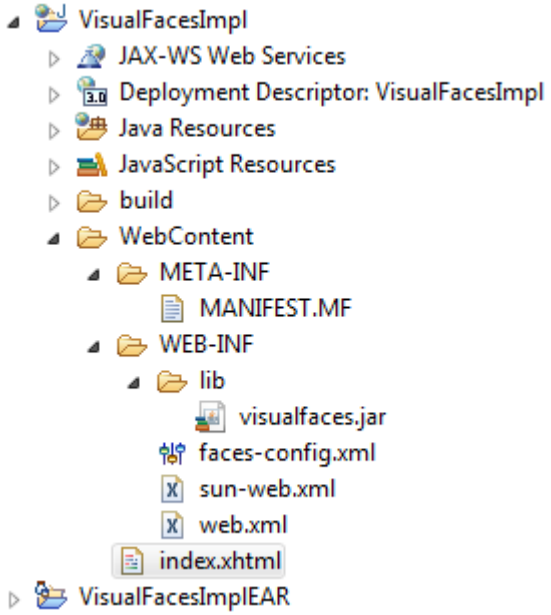

Illustration 3: Structure of the Dynamic Web Project to test the component