ABOUT VEERAMANI KALYANASUNDARAM

Veera is a Software Architect working in telecom domain with rich experience in Java Middleware Technologies. He is a OOAD practitioner and interested in Performance Engineering.

JSF Components Listeners Example

Posted by: Veeramani Kalyanasundaram in jsf March 25th, 2015

In this example of JSF Components Listeners, we will discuss about various component listeners provided by Java Server Faces and show you different ways of using the listeners.

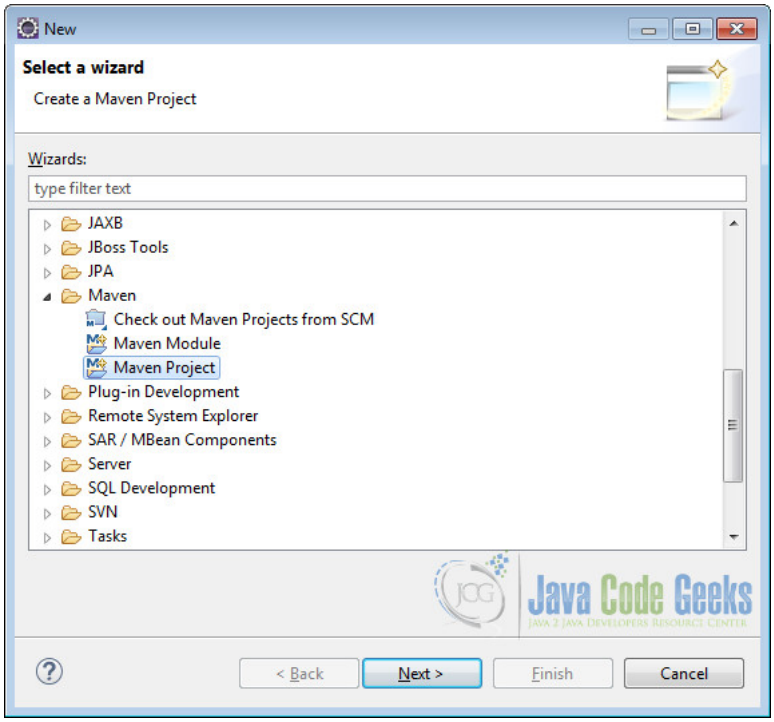
In a web page when the user makes changes to the input component or performs an action on the UI component, the JSF fires an event. These events can be handled by application to take necessary action. JSF provides listeners to capture the event. We can implement the listeners as classes or use the backing bean method to capture the event. Depending upon how the listener is implemented, the page can either use listener tag or listener attribute of the UI component. We will show you both the approaches here. Let's begin with setting up a JSF project and do all the necessary configuration to run the application.

Our preferred environment is Eclipse. We are using Eclipse Luna SR1 with Maven Integration Plugin, JDK 8u25 (1.8.0_25) and Tomcat 8 application server. Having said that, we have tested the code against JDK 1.7 and Tomcat 7 as well.

Tip
You may skip project creation and jump directly to the **beginning of the example** below.

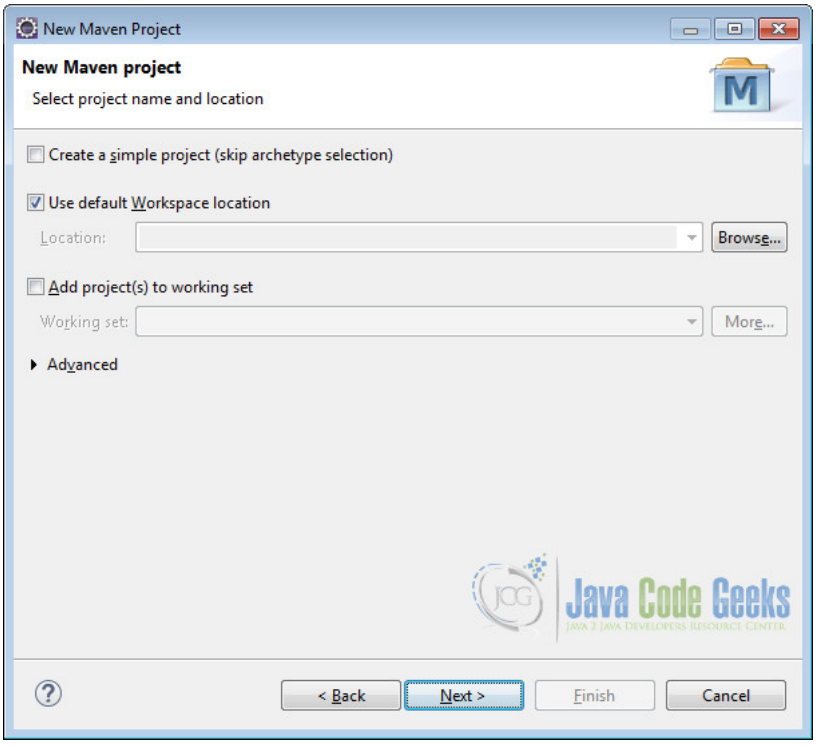
1. Create a new Maven Project

Go to File -> New->Other-> Maven Project



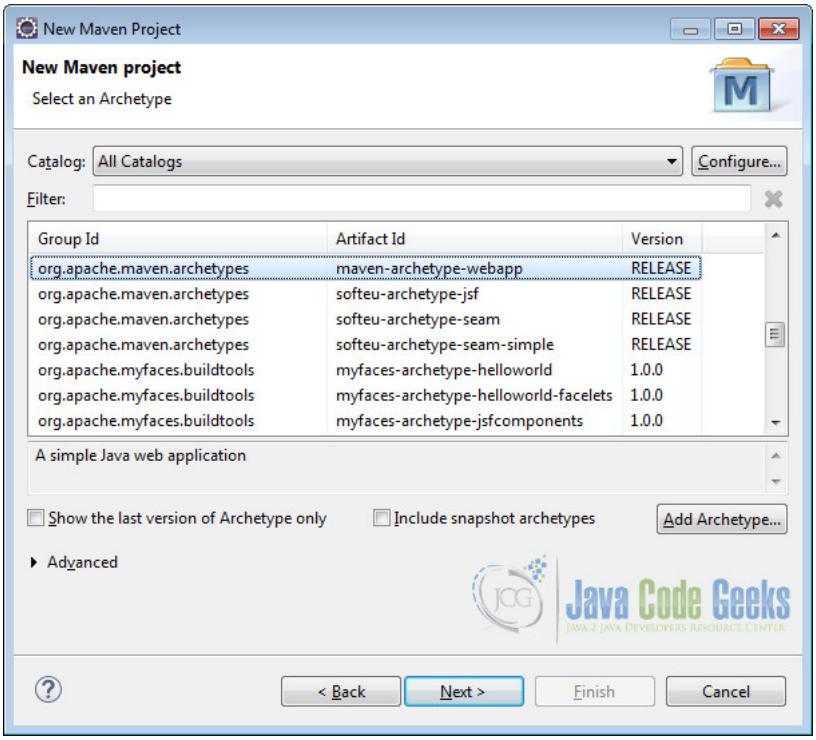
Maven setup – step 1

In the "Select project name and location" page of the wizard, make sure that "Create a simple project (skip archetype selection)" option is **unchecked**, hit "Next" to continue with default values.



Maven setup – step 2

Here choose "maven-archetype-webapp" and click on Next.



Maven setup – step 3

In the "Enter an artifact id" page of the wizard, you can define the name and main package of your project. Set the "Group Id" variable to

```
"com.javacodegeeks.snippets.enterprise"
```

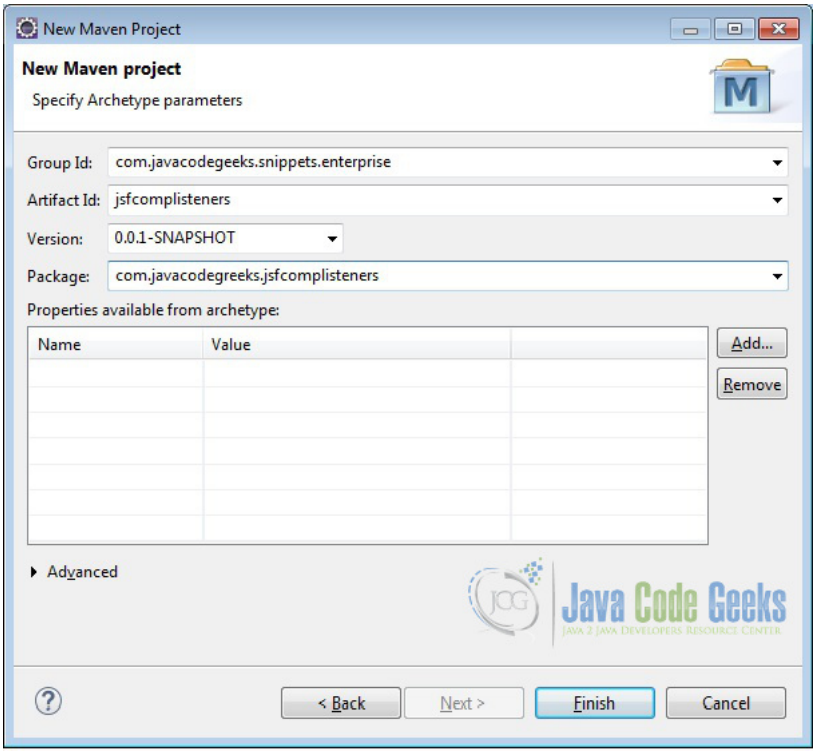
and the "Artifact Id" variable to

```
"jsfcomplisteners"
```

. For package enter

```
"com.javacodegreeks.jsfcomplisteners"
```

and hit "Finish" to exit the wizard and to create your project.

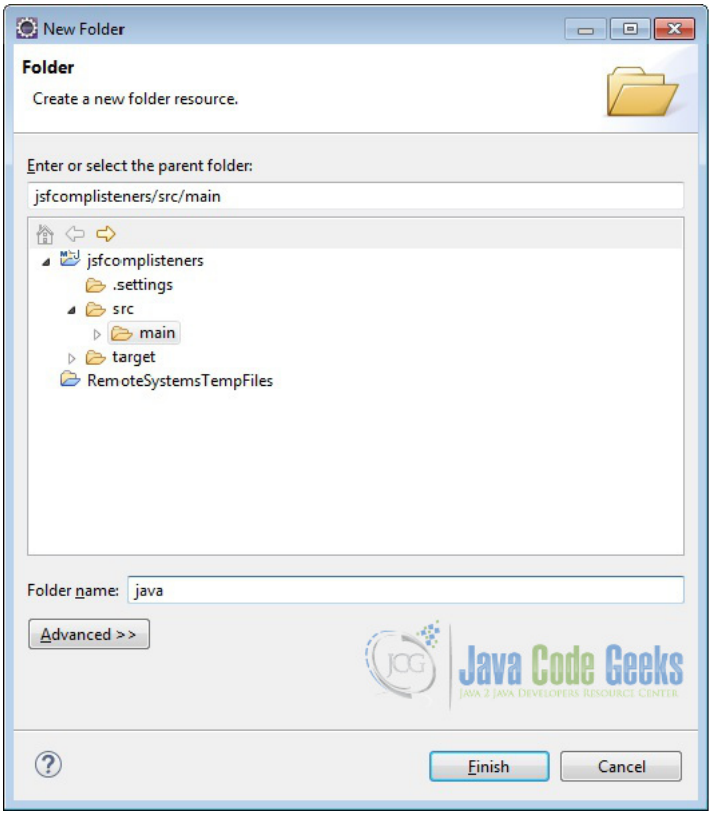


Maven setup – step 4

Now create a folder called java under

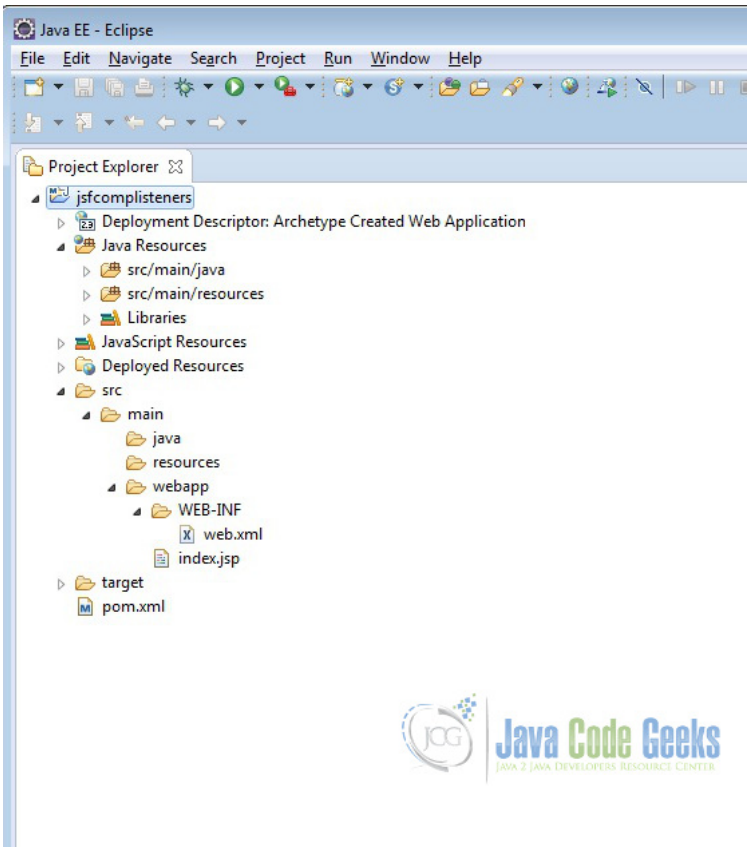
```
src/main
```

.



Maven setup – step 5

Refresh the project. Finally, the project structure will look like the below.

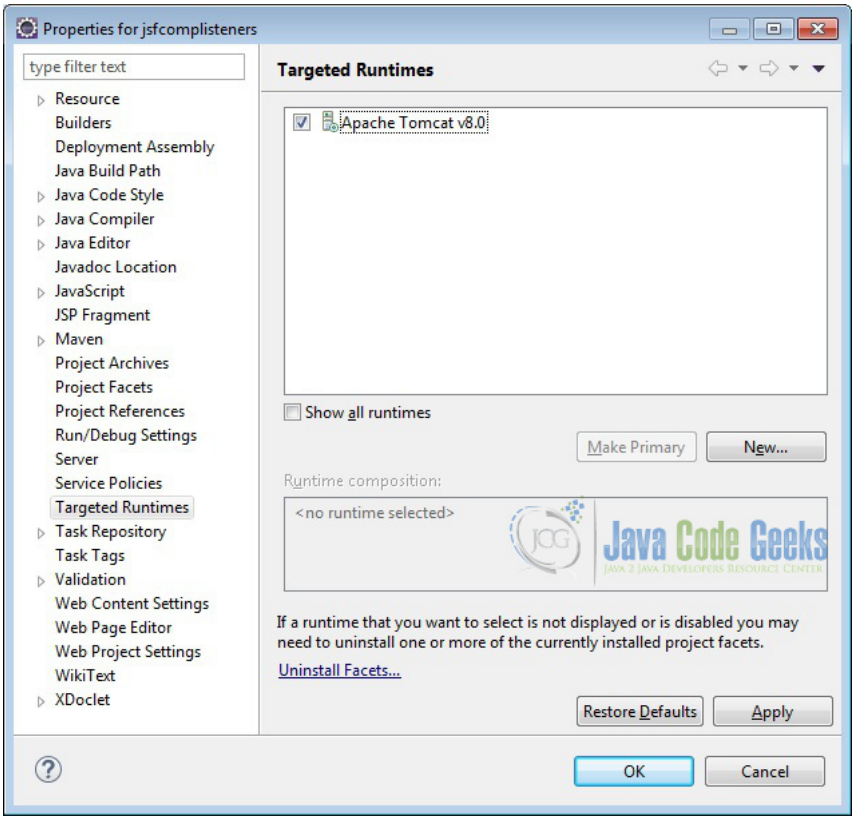


Maven setup – step 6

If you see any errors in the

```
index.jsp
```

, set target runtime for the project.



Maven setup – step 7

2. Modify POM to include JSF dependency

Add the dependencies in Maven's

`pom.xml`

file, by editing it at the "Pom.xml" page of the POM editor.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.javacodegreeks.snippets.enterprise</groupId>
 <artifactId>jsfcomplisteners</artifactId>
 <packaging>war</packaging>
 <version>0.0.1-SNAPSHOT</version>
 <name>jsfcomplisteners Maven Webapp</name>
 <url>http://maven.apache.org</url>
 <dependencies>
   <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
   </dependency>
   <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.2.9</version>
   </dependency>
   <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.2.9</version>
   </dependency>
 </dependencies>
 <build>
 <finalName>jsfcomplisteners</finalName>
 </build>
</project>
```

3. Add Faces Servlet in web.xml

The

`web.xml`

file has to be modified to include the faces servlet configuration as below.

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
 <display-name>Archetype Created Web Application</display-name>
 <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>*.xhtml</url-pattern>
  </servlet-mapping>
</web-app>
```

4. Value Change Listener

JSF input components fires value-change-event when the user interacts with them. The input components such as

```
h:inputText
```

or

```
h:selectOneMenu
```

fire the value-change-event event on modifying the component values. JSF provides two mechanism for implementing the listeners. We will show you how to implement value change listener on

```
h:selectOneMenu
```

by using both the techniques.

First, lets create a package called

```
com.javacodegeeks.jsfcomplisteners
```

under the folder

```
src/main/java
```

. Now we create a managed bean called

```
JavaVersion
```

. The

```
@ManagedBean
```

annotation makes the POJO as managed bean. The

```
@SessionScoped
```

annotation on the bean makes the bean available to the entire user session. We will use

```
java.util.LinkedHashMap
```

to store the Java versions released along with the date of release.

JavaVersion.java

```
package com.javacodegeeks.jsfcomplisteners;

import java.util.LinkedHashMap;
import java.util.Map;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "javaVersion", eager = true)
@SessionScoped
public class JavaVersion {

 private static Map<String, String> versionData;
 private String releaseDate = "January 23, 1996";

 static {
 versionData = new LinkedHashMap<String, String>();
 versionData.put("JDK 1.0", "January 23, 1996");
 versionData.put("JDK 1.1", "February 19, 1997");
 versionData.put("J2SE 1.2", "December 8, 1998");
 versionData.put("J2SE 1.3", "May 8, 2000");
 versionData.put("J2SE 1.4", "February 6, 2002");
 versionData.put("J2SE 5.0", "September 30, 2004");
 versionData.put("Java SE 6", "December 11, 2006");
 versionData.put("Java SE 7", "July 28, 2011");
 versionData.put("Java SE 8", "March 18, 2014");
 }

 public Map<String, String> getVersionData() {
  return versionData;
 }

 public void setVersionData(Map<String, String> versionData) {
  JavaVersion.versionData = versionData;
 }

 public String getReleaseDate() {
  return releaseDate;
 }

 public void setReleaseDate(String releaseDate) {
  this.releaseDate = releaseDate;
 }

}
```

4.1 Using valueChangeListener attribute

To use the UI component valueChangeListener attribute technique, we need to first create a bean method. Let's modify the

```
JavaVersion
```

backing bean to include the listener method.

JavaVersion.java

```
01 package com.javacodegeeks.jsfcomplisteners;
02
03 import java.util.LinkedHashMap;
04 import java.util.Map;
05
06 import javax.faces.bean.ManagedBean;
07 import javax.faces.bean.SessionScoped;
08 import javax.faces.event.ValueChangeEvent;
09
10 @ManagedBean(name = "javaVersion", eager = true)
11 @SessionScoped
12 public class JavaVersion {
13
14  private static Map<String, String> versionData;
15  private String releaseDate = "January 23, 1996";
16
17  static {
18  versionData = new LinkedHashMap<String, String>();
19  versionData.put("JDK 1.0", "January 23, 1996");
20  versionData.put("JDK 1.1", "February 19, 1997");
21  versionData.put("J2SE 1.2", "December 8, 1998");
22  versionData.put("J2SE 1.3", "May 8, 2000");
23  versionData.put("J2SE 1.4", "February 6, 2002");
24  versionData.put("J2SE 5.0", "September 30, 2004");
25  versionData.put("Java SE 6", "December 11, 2006");
26  versionData.put("Java SE 7", "July 28, 2011");
27  versionData.put("Java SE 8", "March 18, 2014");
28  }
29
30  public Map<String, String> getVersionData() {
31   return versionData;
32  }
33
34  public void setVersionData(Map<String, String> versionData) {
35   JavaVersion.versionData = versionData;
36  }
37
38  public String getReleaseDate() {
39   return releaseDate;
40  }
41
42  public void setReleaseDate(String releaseDate) {
43   this.releaseDate = releaseDate;
44  }
45
46  public void versionChanged(ValueChangeEvent event) {
47   releaseDate = event.getNewValue().toString();
48  }
49
50 }
```

Now we will create a view called

```
attrlistener.xhtml
```

under

```
/src/main/webapp
```

. We have used

```
h:selectOneMenu
```

to display various Java releases and used

```
h:outputText
```

to display the release date. We will use the

```
valueChangeListener
```

attribute of

```
h:selectOneMenu
```

to invoke the bean method.

attrlistener.xhtml

```
01 <?xml version="1.0" encoding="ISO-8859-1" ?>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04  xmlns:h="http://java.sun.com/jsf/html"
05  xmlns:f="http://java.sun.com/jsf/core">
06 <head>
07 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
08 <title>Listener Attribute</title>
09 </head>
10 <body>
11  <h:form>
```

```
<div>
 <h3>Using valueChangeListener attribute</h3>
 <div>
  <strong>Selected Version : </strong>
  <h:selectOneMenu value="#{javaVersion.releaseDate}"
  onchange="submit()"
  valueChangeListener="#{javaVersion.versionChanged}">
  <f:selectItems value="#{javaVersion.versionData}" />
  </h:selectOneMenu>
 </div>
 <br />
 <div>
  <strong>Release Date : </strong>
  <h:outputText value="#{javaVersion.releaseDate}" />
 </div>
</div>
</h:form>
</body>
</html>
```

Now we can create the deployment package using Run as –> Maven clean and then Run as –> Maven install. This will create a war file in the target folder. The

```
war
```

file produced must be placed in

```
webapps
```

folder of tomcat. Now we can start the server.

Open the following URL in the browser.

```
http://localhost:8080/jsfcomplisteners/attrlistener.xhtml
```

valueChangeListener – Attribute

Modify the Java version using the drop down. The release date will get changed accordingly.

4.2 Using valueChangeListener Tag

To use the valueChangeListener Tag, we need to first create a class implementing the

```
valueChangeListener
```

interface. Lets first create a class called

```
VersionChangeListener
```

in the package

```
com.javacodegeeks.jsfcomplisteners
```

and implement the

```
processValueChange
```

method of the interface. We will use the

```
FacesContext
```

to get the

```
JavaVersion
```

object and set the release date using the

```
ValueChangeEvent
```

.

VersionChangeListener.java

```
01 package com.javacodegeeks.jsfcomplisteners;
02
03 import javax.faces.context.FacesContext;
04 import javax.faces.event.AbortProcessingException;
05 import javax.faces.event.ValueChangeEvent;
06 import javax.faces.event.ValueChangeListener;
07
08 public class VersionChangeListener implements ValueChangeListener{
09
10  public void processValueChange(ValueChangeEvent event)
11   throws AbortProcessingException {
12    JavaVersion javaVersion= (JavaVersion) FacesContext.getCurrentInstance().
13     getExternalContext().getSessionMap().get("javaVersion");
14    javaVersion.setReleaseDate(event.getNewValue().toString());
15
16  }
17
18 }
```

We will create a view called

```
taglistener.xhtml
```

under

```
/src/main/webapp
```

. We have used

```
h:selectOneMenu
```

to display various Java releases and used

```
h:outputText
```

to display release date. But instead of using the valueChangeListener attribute, we are using the tag

```
f:valueChangeListener
```

this time. The

```
type
```

attribute of the tag reference to the fully qualified name of the listener which is “

```
com.javacodegeeks.jsfcomplisteners.VersionChangeListener
```

” in our case.

taglistener.xhtml

```
01 <?xml version="1.0" encoding="ISO-8859-1" ?>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04  xmlns:h="http://java.sun.com/jsf/html"
05  xmlns:f="http://java.sun.com/jsf/core">
06 <head>
07 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
08 <title>Listener Tag</title>
09 </head>
10 <body>
11  <h:form>
12    <div>
13     <h3>Using valueChangeListener Tag</h3>
14     <div>
15      <strong>Selected Version : </strong>
16      <h:selectOneMenu value="#{javaVersion.releaseDate}"
17       onchange="submit()">
18      <f:valueChangeListener
19       type="com.javacodegeeks.jsfcomplisteners.VersionChangeListener" />
20      <f:selectItems value="#{javaVersion.versionData}" />
21      </h:selectOneMenu>
22     </div>
23     <br />
24     <div>
25      <strong>Release Date : </strong>
26      <h:outputText value="#{javaVersion.releaseDate}" />
27     </div>
28    </div>
29  </h:form>
30 </body>
31 </html>
```

Now again package using maven and copy the

```
war
```

to the apache tomcat

```
webapps
```

folder. Open the following URL in the browser.

```
http://localhost:8080/jsfcomplisteners/taglistener.xhtml
```

valueChangeListener – Tag

Modify the Java version using the drop down. The release date will get changed accordingly.

5. Action Listener

JSF UI components fires the action-event when the user clicks on them. The components such as h:commandButton fire the event on click of it. Similar to

`valueChangeListener`

,

`actionListener`

can also be implemented in two techniques. We will show you how to implement action listener on

`h:commandButton`

using both the techniques.

Now, we create a managed bean called

`UserProfile`

under the package

`com.javacodegeeks.jsfcomplisteners`

. The

`@ManagedBean`

annotation makes the POJO as managed bean. The

`@SessionScoped`

annotation on the bean makes the bean available to the entire user session. We will use the method

`updateGreeting`

to modify the greeting with current day. A utility method called

`getDayUtil`

is also provided to convert the day into user readable value.

UserProfile.java

```
01 package com.javacodegeeks.jsfcomplisteners;
02
03 import java.util.Calendar;
04
05 import javax.faces.bean.ManagedBean;
06 import javax.faces.bean.SessionScoped;
07 import javax.faces.event.ActionEvent;
08
09 @ManagedBean(name = "userProfile", eager = true)
10 @SessionScoped
11 public class UserProfile {
12
13  private String label = "Click here for Today's Greeting ";
14  private String greeting = "Hello, have a nice";
15
16  public String getGreeting() {
17   return greeting;
18  }
19
20  public String getLabel() {
21  return label;
22  }
23
```

```
24  public void setLabel(String label) {
25   this.label = label;
26  }
27
28  public void setGreeting(String greeting) {
29   this.greeting = greeting;
30  }
31
32  public void updateGreeting(ActionEvent event) {
33   greeting = greeting + " "
34     + getDayUtil(Calendar.getInstance().get(Calendar.DAY_OF_WEEK))
35     + "!!";
36  }
37
38  private String getDayUtil(int day) {
39    String dayOfWeek = "Sunday";
40    switch (day) {
41     case 1:
42      dayOfWeek = "Sunday";
43      break;
44     case 2:
45      dayOfWeek = "Monday";
46      break;
47     case 3:
48      dayOfWeek = "Tuesday";
49      break;
50     case 4:
51      dayOfWeek = "Wednesday";
52      break;
53     case 5:
54      dayOfWeek = "Thursday";
55      break;
56     case 6:
57      dayOfWeek = "Friday";
58      break;
59     case 7:
60      dayOfWeek = "Saturday";
61      break;
62    }
63   return dayOfWeek;
64  }
65
66 }
```

5.1 Using actionListener attribute

To use the UI component actionListener attribute technique, we will use the backing bean method

```
updateGreeting
```

of UserProfile.

Now lets modify the view

```
attrlistener.xhtml
```

to include

```
h:outputText
```

for creating label and

```
h:commandButton
```

for command button. The

```
actionListener
```

attribute of

```
h:commandButton
```

is used to invoke the backing bean method.

attrlistener.xhtml

```
01 <?xml version="1.0" encoding="ISO-8859-1" ?>
02 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
03 <html xmlns="http://www.w3.org/1999/xhtml"
04  xmlns:h="http://java.sun.com/jsf/html"
05  xmlns:f="http://java.sun.com/jsf/core">
06 <head>
07 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
08 <title>Listener Attribute</title>
09 </head>
10 <body>
11  <h:form>
12   <div>
13    <h3>Using valueChangeListener attribute</h3>
14    <div>
15     <strong>Selected Version : </strong>
16     <h:selectOneMenu value="#{javaVersion.releaseDate}"
17      onchange="submit()"
18      valueChangeListener="#{javaVersion.versionChanged}">
19     <f:selectItems value="#{javaVersion.versionData}" />
```



```
24     <strong>Release Date : </strong>
25     <h:outputText value="#{javaVersion.releaseDate}" />
26    </div>
```

```
  </div>

  <hr></hr>
  <div>
   <h3>Using actionListener attribute</h3>
   <div>
    <h:outputText value="#{userProfile.label}"></h:outputText>
    <h:commandButton id="submit" value="Submit" action="greeting"
    actionListener="#{userProfile.updateGreeting}" />
   </div>
  </div>
 </h:form>
</body>
</html>
```

Now create another view called

```
greeting.xhtml
```

under

```
/src/main/webapp
```

. By means of Implicit navigation the action value of the commandButton in

```
attrlistener.xhtml
```

will get resolved to

```
greeting.xhtml
```

. This page is a simple page to display the updated greeting.

greeting.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:h="http://java.sun.com/jsf/html"
 xmlns:f="http://java.sun.com/jsf/core">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Result</title>
</head>
<body>
 <h:form>
   <div>
    <h3>Action Listener Result Page</h3>
    <div>
     <h:outputText value="#{userProfile.greeting}" />
    </div>
   </div>
 </h:form>
</body>
</html>
```

Now again package using maven and copy the

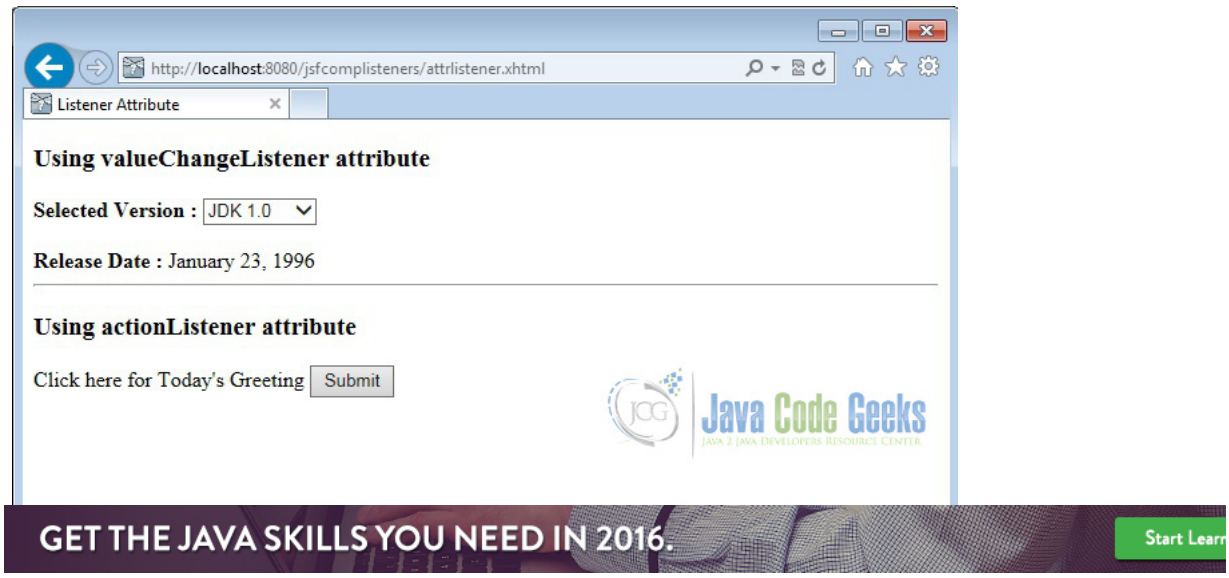
```
war
```

to the apache tomcat

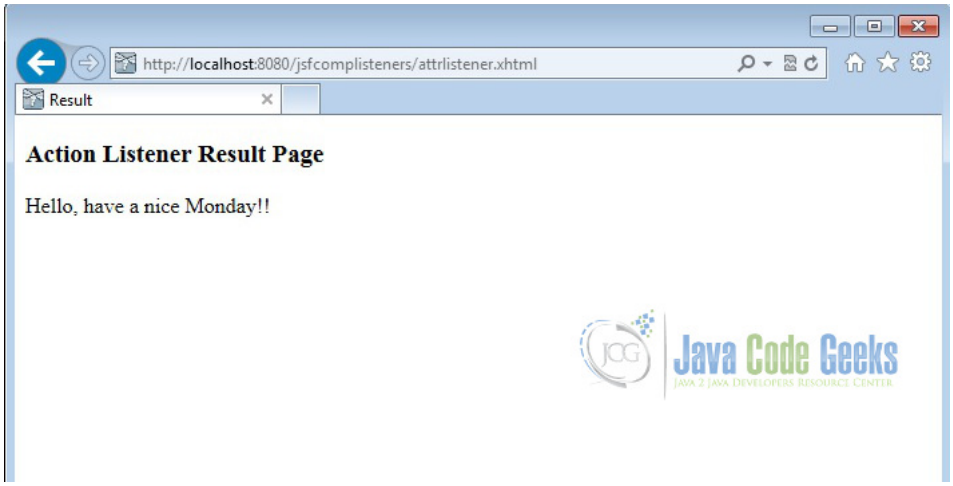
```
webapps
```

folder. Open the following URL in the browser.

```
http://localhost:8080/jsfcomplisteners/attrlistener.xhtml
```



Now click on the submit button.



actionListener – Attribute Result

5.2 Using actionListener Tag

To use the

`actionListener`

Tag, we need to first create a class implementing the

`ActionListener`

interface. Lets first create a class called

`UserProfileActionListener`

in the package

`com.javacodegeeks.jsfcomplisteners`

which implement the method

`processAction`

of the interface. We will use the

`FacesContext`

to get the

`UserProfile`

object and update the greeting using

`ActionEvent`

.

UserProfileActionListener.java

```
01 package com.javacodegeeks.jsfcomplisteners;
02
03 import javax.faces.context.FacesContext;
04 import javax.faces.event.AbortProcessingException;
05 import javax.faces.event.ActionEvent;
06 import javax.faces.event.ActionListener;
07
08 public class UserProfileActionListener implements ActionListener {
09
10  public void processAction(ActionEvent event)
11   throws AbortProcessingException {
12    UserProfile userProfile = (UserProfile) FacesContext
13     .getCurrentInstance().getExternalContext().getSessionMap()
14     .get("userProfile");
15    userProfile.updateGreeting(event);
16  }
17 }
```

Now let us modify the view

`taglistener.xhtml`

to include

`h:outputText`

for creating label and

for command button. We will use the

`f:actionListener`

tag and reference the

```
type
```

attribute to the fully qualified name of the class

```
com.javacodegeeks.jsfcomplisteners.UserProfileActionListener
```

.

taglistener.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:h="http://java.sun.com/jsf/html"
 xmlns:f="http://java.sun.com/jsf/core">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Listener Tag</title>
</head>
<body>
 <h:form>
  <div>
   <h3>Using valueChangeListener Tag</h3>
   <div>
    <strong>Selected Version : </strong>
    <h:selectOneMenu value="#{javaVersion.releaseDate}"
     onchange="submit()">
    <f:valueChangeListener
     type="com.javacodegeeks.jsfcomplisteners.VersionChangeListener" />
    <f:selectItems value="#{javaVersion.versionData}" />
    </h:selectOneMenu>
   </div>
   <br />
   <div>
    <strong>Release Date : </strong>
    <h:outputText value="#{javaVersion.releaseDate}" />
   </div>
  </div>
  <hr></hr>
  <div>
   <h3>Using actionListener Tag</h3>
   <div>
    <h:outputText value="#{userProfile.label}"></h:outputText>
    <h:commandButton id="submit" value="Submit" action="greeting">
    <f:actionListener
     type="com.javacodegeeks.jsfcomplisteners.UserProfileActionListener" />
    </h:commandButton>
  </div>
 </div>
 </h:form>
</body>
</html>
```

Now again package using maven and copy the

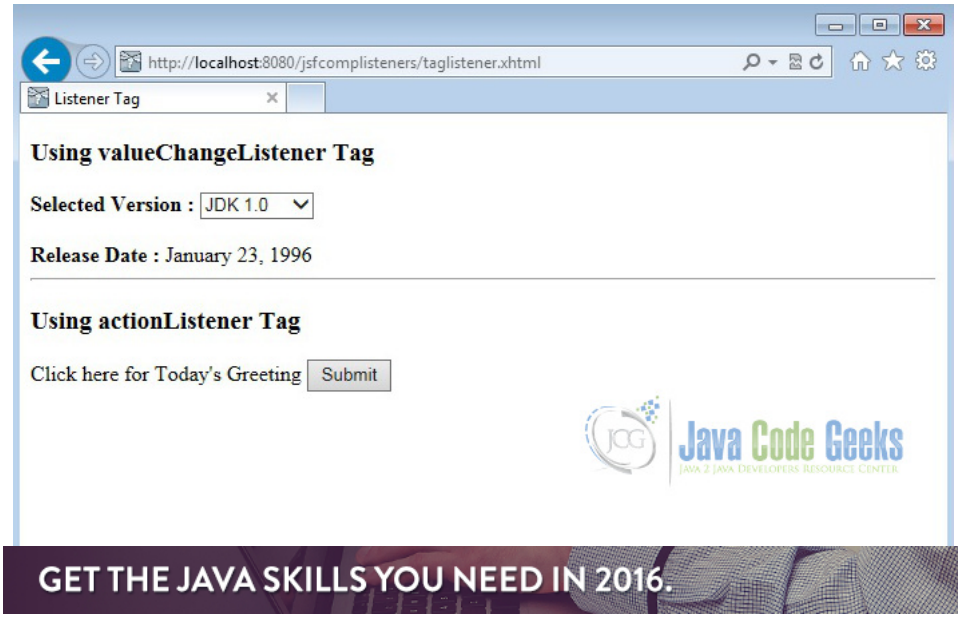
```
war
```

to the apache tomcat

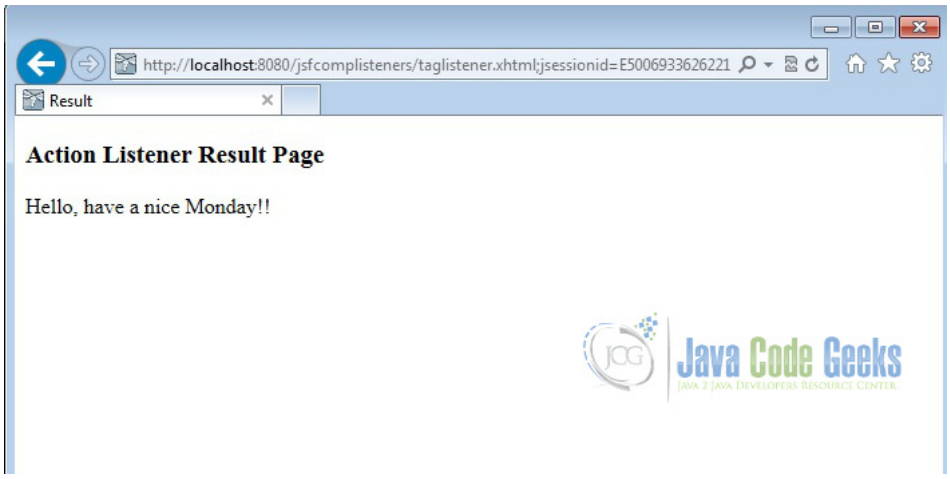
```
webapps
```

folder. Open the following URL in the browser

```
http://localhost:8080/jsfcomplisteners/taglistener.xhtml
```



Now Click on the Submit button.



actionListener – Tag Result

6. Download the Eclipse Project

This was an example of how to use Java Server Faces Component Listeners.

Download
You can download the full source code of this example here : **JSF Components Listeners**

Tagged with: JSF LISTENERS

Whitepapers

THE CODE GEEKS NETWORK

.NET Code Geeks

Java Code Geeks

System Code Geeks

Web Code Geeks

Java write to File Example

java.io.FileNotFoundException – How to solve File Not Found Exception

java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception

java.lang.NoClassDefFoundError – How to solve No Class Def Found Error

JSON Example With Jersey + Jackson

Spring JdbcTemplate Example

DISCLAIMER