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 Standard Converters Example

Posted by: Veeramani Kalyanasundaram in jsf March 16th, 2015

In this example of JSF Standard Converters, we will show you how Java Server Faces standard converters work and also discuss the various options available to use standard converters.

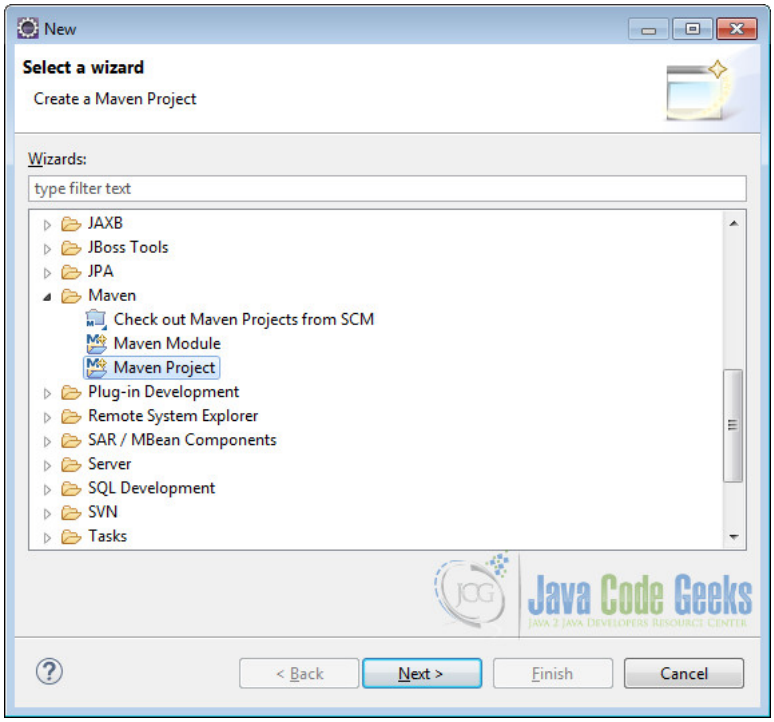
When a request is sent from the browser to the application server, the form values are sent as String. To convert the String into Java Objects, we need to use converters. Similarly when the Java Object is passed back from the application server and gets converted into HTML, we need to convert it to String. JSF provides a set of standard converters as well as give option to create custom converters. 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.

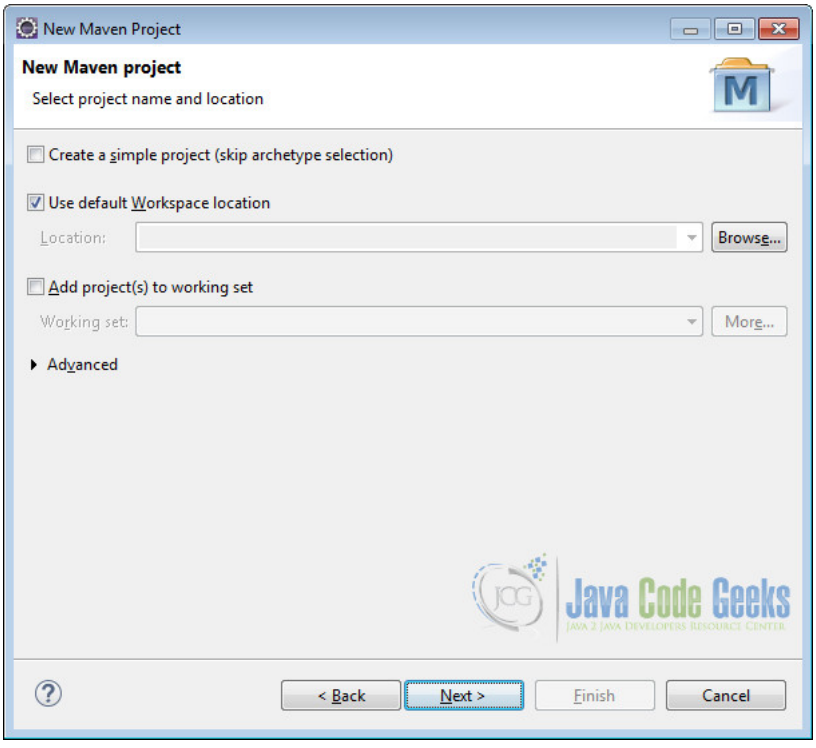
1. Create a new Maven Project

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



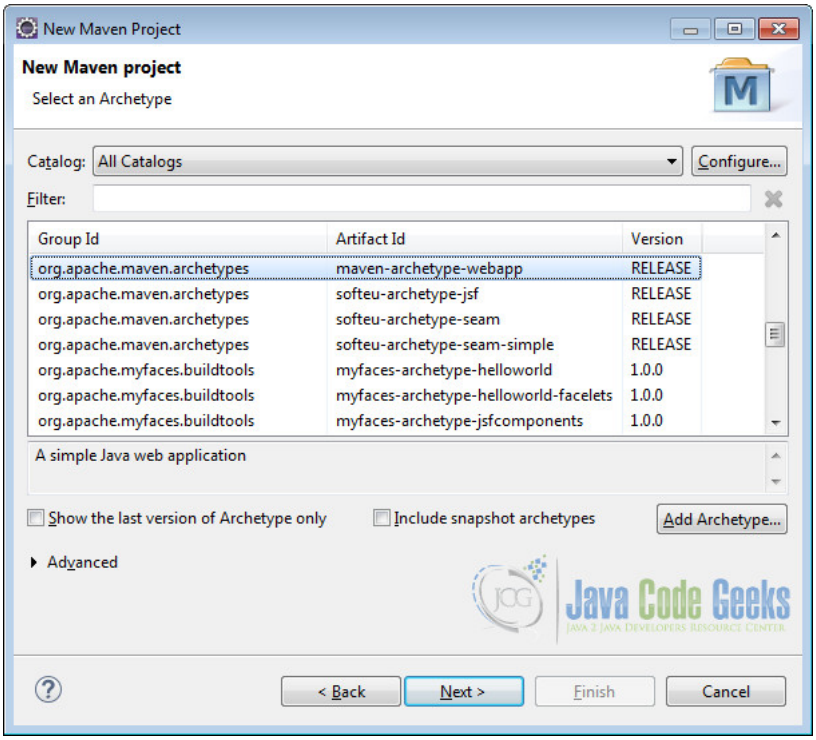
Maven Setup – Step 1

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"
```

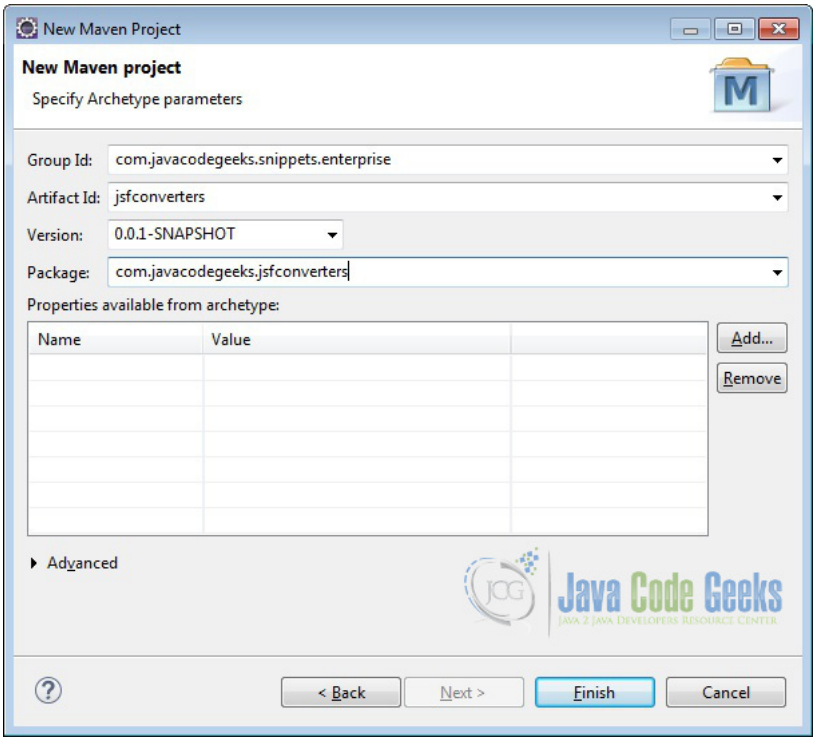
and the "Artifact Id" variable to

```
"jsfconverters"
```

. For package enter

```
"com.javacodegreeks.jsfconverters"
```

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

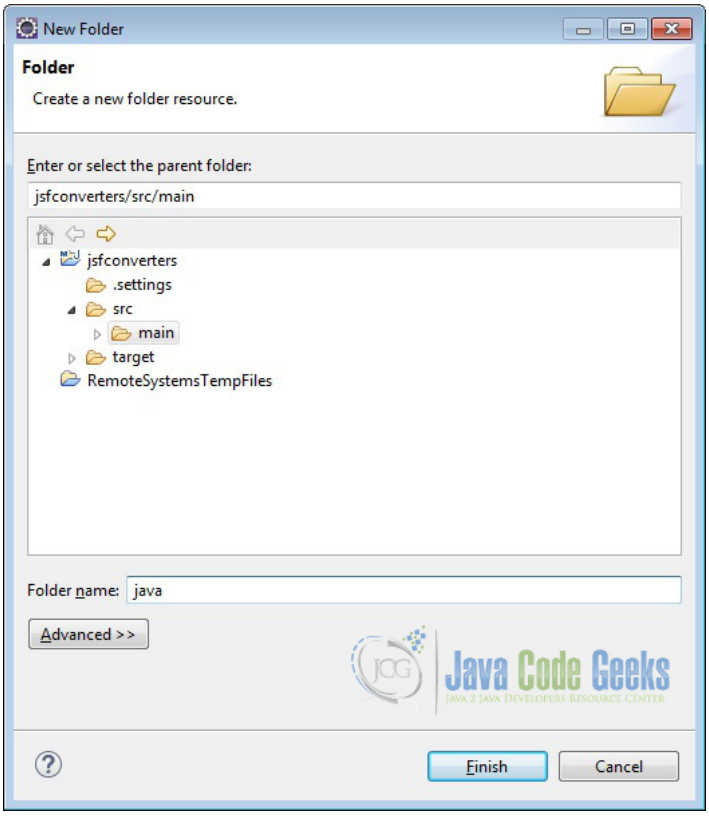


Maven setup – step 4

Now create a folder called java under

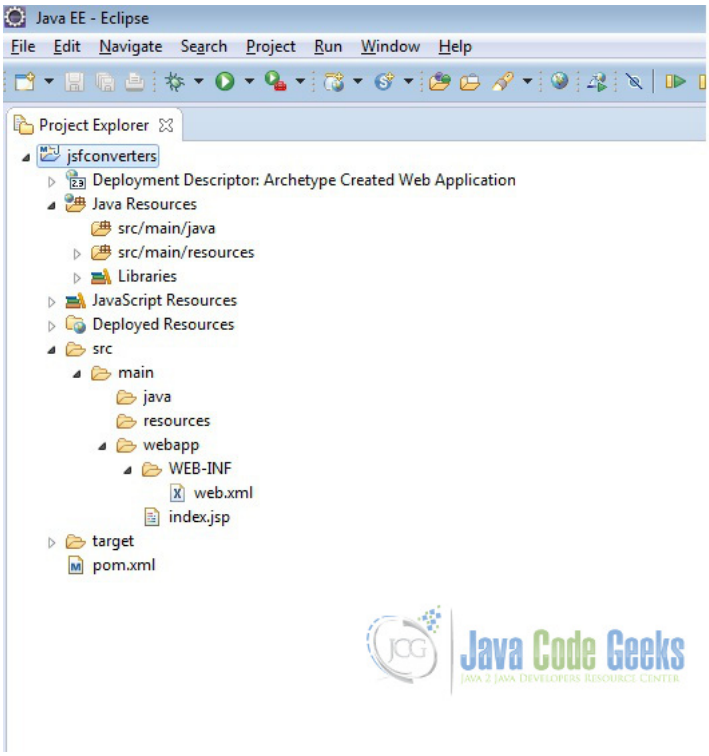
```
src/main
```

.



Maven setup – step 5

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

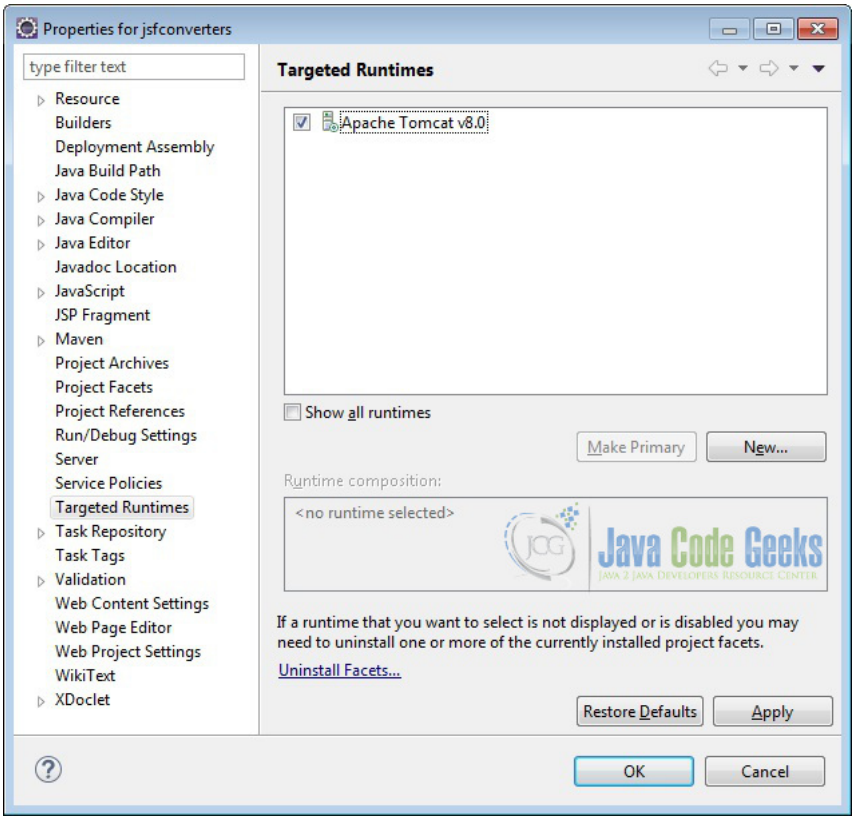


Maven setup – step 6

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.

pom.xml

```
<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.javacodegeeks.snippets.enterprise</groupId>
    <artifactId>jsfconverters</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>jsfconverters 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>jsfconverters</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.

web.xml

```
<!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. Standard Converters

JSF provides a set of standard converters. The following section shows all the available converters and state the purpose for each of the converter.

- **BigDecimalConverter** – For Conversion between String and
  `java.math.BigDecimal`
- **BigIntegerConverter** – For Conversion between String and
  `java.math.BigInteger`
- **BooleanConverter** – For Conversion between String and
  `java.lang.Boolean`
- **ByteConverter** – For Conversion between String and
  `java.lang.Byte`
- **CharacterConverter** – For Conversion between String and
  `java.lang.Character`
- **DateTimeConverter** – For Conversion between String and
  `java.util.Date`
- **DoubleConverter** – For Conversion between String and
  `java.lang.Double`
- **FloatConverter** – For Conversion between String and
  `java.lang.Float`
- **IntegerConverter** – For Conversion between String and
  `java.lang.Integer`
- **LongConverter** – For Conversion between String and
  `java.lang.Long`
- **NumberConverter** – For Conversion between String and
  `java.lang.Number`
- **ShortConverter** – For Conversion between String and
  `java.lang.Short`

The **DateTimeConverter** and **NumberConverter** have their own tags and provide attributes for data conversion. We will discuss about the two converters in detail later.

5. How to use Converters

JSF provides three ways to use converters. We can use any of the method depending on the type of converter.

5.1 Using converter attribute

We can add the

`converter`

attribute to the UI component using the fully qualified class name.

```
<h:inputText id="age" converter="javax.faces.Integer" />
```

5.2 Using f:converter tag

We can include the

`f:converter`

tag within a component. The

```
converterID
```

attribute point to reference the converter's name.

```
<h:inputText id="age">
     <f:converter converterID="javax.faces.Integer" />
</h:inputText>
```

5.3 Using converter tags

We can use the standard converter tags provided in the JSF.

```
<h:outputText value="#{userBean.height}">
         <f:convertNumber maxFractionDigits="2" />
</h:outputText>
```

Or by using custom converter

```
<h:outputText value="#{userBean.ssn}">
         <my:ssnConverter country="US" />
</h:outputText>
```

6. Implicit Converter

JSF provides standard converters that automatically perform conversion for basic Java types. We will see it working by creating an example.

First we will create a package called

```
com.javacodegeeks.jsfconverters
```

under the folder

```
src/main/java
```

. Now, we need to create a managed bean called

```
UserBean
```

. The

```
@ManagedBean
```

annotation makes the POJO as managed bean. The

```
@SessionScoped
```

annotation on the bean makes the bean available to the entire user session.

UserBean.java

```
package com.javacodegeeks.jsfconverters;

import java.util.Date;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="userBean", eager=true)
@SessionScoped
public class UserBean {

    private String firstName;
    private String lastName;
    private int age;
    private Date dateOfBirth;
    private Double height;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

        public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

```

```
    public Double getHeight() {
        return height;
    }

    public void setHeight(Double height) {
        this.height = height;
    }


}
```

Now, we will create a view called

```
adduser.xhtml
```

under

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

. We have used

```
h:outputLabel
```

to display the label and

```
h:inputText
```

to get the user input. We will submit the form using the component

```
h:commandButton
```

. The action tag of

```
h:commandButton
```

is set to "viewuser". We will use the implicit navigation feature of JSF for navigating to the "viewuser" page.

adduser.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>Add User</title>
</head>
<body>
    <h:form>
        <h2>Add User</h2>
        <div>
            <h:outputLabel for="firstName">First Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="firstName" label="First Name"
                value="#{userBean.firstName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="lastName">Last Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="lastName" label="Last Name"
                value="#{userBean.lastName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="age">Age</h:outputLabel>
        </div>
        <div>
            <h:inputText id="age" label="age" value="#{userBean.age}">
            </h:inputText>
        </div>
        <div> </div>
        <div> </div>
        <div>
            <h:commandButton value="Submit" action="viewuser"></h:commandButton>
        </div>
    </h:form>
</body>
</html>
```

We will create another view called

```
viewuser.xhtml
```

under

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

to display the values entered by the user.

viewuser.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>View User</title>

</head>
<body>
    <h:form>
        <h2>View User</h2>
        <h4>
            <h:outputText value="#{userBean.firstName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.lastName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.age}"></h:outputText>
        </h4>
    </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
```

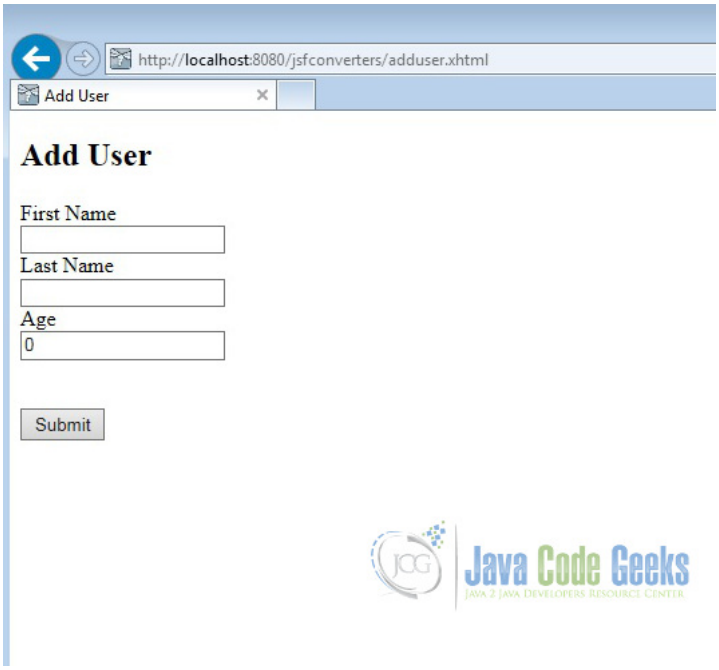
file produced must be placed in

```
webapps
```

folder of tomcat. Now we can start the server.

Open the following URL in the browser.

```
http://localhost:8080/jsfconverters/adduser.xhtml
```

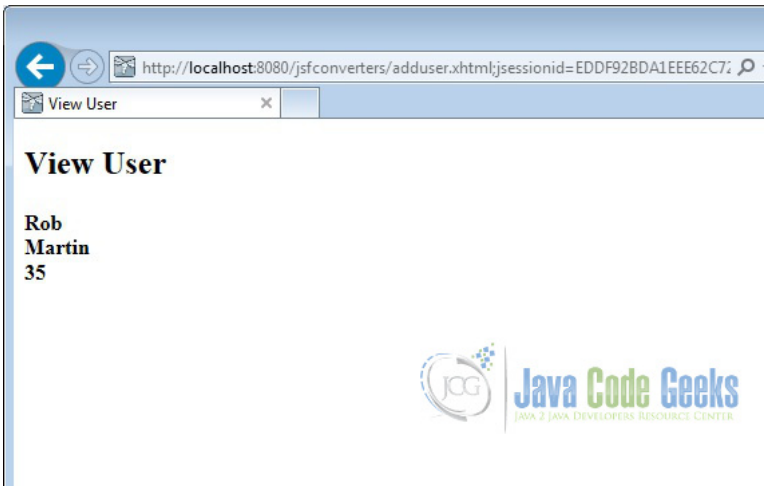


Add User -1

Enter the values for First Name, Last Name and age. Now, click on submit button. JSF will use the implict navigation and display the

```
viewuser.xhtml
```

. Here the value for age is converted to int by the standard converter automatically.



View User – 1

To validate the implicit converter, try to enter **some characters** in the age field and click on the submit. You will see error in **Tomcat server window**.

7. DateTimeConverter

The JSF DateTimeConverter provides the following attributes to convert and format the Date.

- **dateStyle** – Predefined formatting style which determines how the date component of a date string is to be formatted and parsed.
- **locale** – Locale to be used during formatting.
- **pattern** – Custom formatting pattern can be used using this attribute.
- **timeStyle** – Predefined formatting style which determines how the time component of a date string is to be formatted and parsed.
- **timeZone** – Time zone in which to interpret any time information in the date String.
- **type** – Specifies date or time or both to be used during formatting.

Now, we modify the

```
adduser.xhtml
```

to accept the Date of Birth as user input.

adduser.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>Add User</title>
</head>
<body>
    <h:form>
        <h2>Add User</h2>
        <div>
            <h:outputLabel for="firstName">First Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="firstName" label="First Name"
                value="#{userBean.firstName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="lastName">Last Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="lastName" label="Last Name"
                value="#{userBean.lastName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="age">Age</h:outputLabel>
        </div>
        <div>
            <h:inputText id="age" label="age" value="#{userBean.age}">
            </h:inputText>
        </div>
        <div>
            <h:outputLabel for="dateOfBirth">Date of Birth (dd-mm-yyyy)</h:outputLabel>
        </div>
        <div>
            <h:inputText id="dateOfBirth" label="Date of Birth" value="#{userBean.dateOfBirth}">
            <f:convertDateTime pattern="dd-mm-yyyy" />
            </h:inputText>
        </div>
        <div> </div>
        <div> </div>
        <div>
            <h:commandButton value="Submit" action="viewuser"></h:commandButton>
        </div>
    </h:form>
</body>
</html>
```

Now modify the

```
viewuser.xhtml
```

to display the date of birth in a different format using the

```
f:convertDateTime
```

tag and its

```
pattern
```

attribute.

viewuser.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>View User</title>

</head>
<body>
    <h:form>
        <h2>View User</h2>
        <h4>
            <h:outputText value="#{userBean.firstName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.lastName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.age}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.dateOfBirth}">
                <f:convertDateTime pattern="dd-MMM-yy"></f:convertDateTime>
            </h:outputText>
        </h4>
    </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/jsfconverters/adduser.xhtml
```

Add User -2

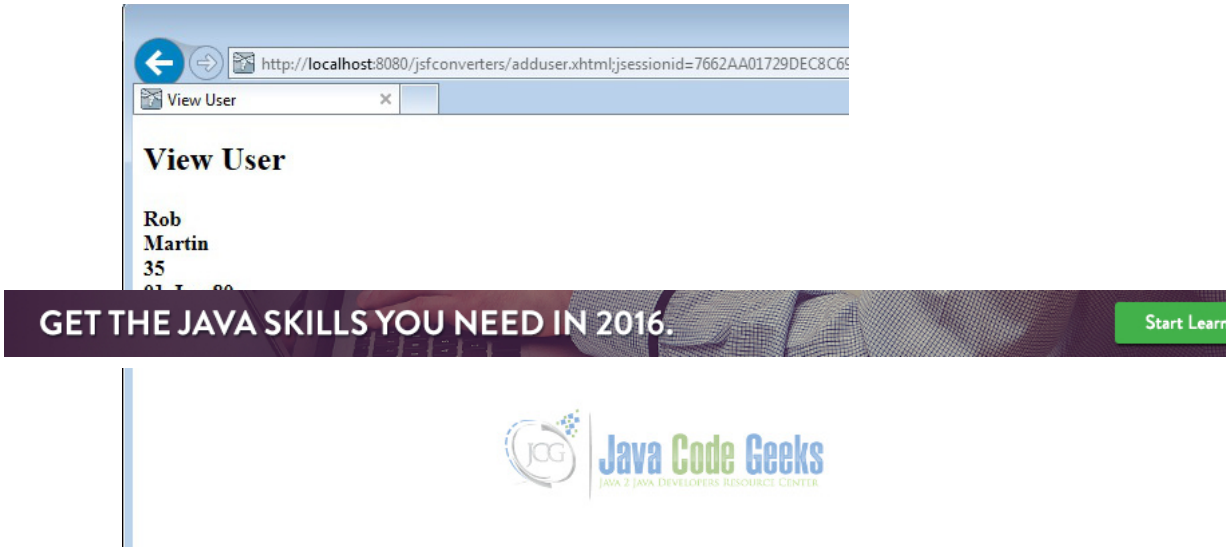
Enter the values for first name, last name, age and date of birth. Now, click on the submit button. The JSF will use the implicit navigation to forward the request to

```
viewuser.xhtml
```

. we will see the date of birth being displayed in the new format

```
dd-MMM-yy
```

defined using the pattern attribute.



View User -2

8. NumberConverter

The JSF NumberConverter provides the following attributes to convert and format the number.

- **currencyCode** – To apply the currency code.
- **currencySymbol** – To apply the currency symbol.
- **groupingUsed** – Flag specifying whether formatted output will contain grouping separators.
- **integerOnly** – Flag specifying whether only the integer part of the value will be formatted and parsed.
- **locale** – Locale whose predefined styles for numbers are used during formatting and parsing.
- **maxFractionDigits** – Maximum number of digits in the fractional portion.
- **maxIntegerDigits** – Maximum number of digits in the integer portion.
- **minFractionDigits** – Minimum number of digits in the fractional portion.
- **minIntegerDigits** – Minimum number of digits in the integer portion.
- **pattern** – To define the custom formatting pattern.
- **type** – Specifies one of number, currency and percent.

Now, we modify the adduser.xhtml to accept the height as user input.

adduser.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>Add User</title>
</head>
<body>
    <h:form>
        <h2>Add User</h2>
        <div>
            <h:outputLabel for="firstName">First Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="firstName" label="First Name"
                value="#{userBean.firstName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="lastName">Last Name</h:outputLabel>
        </div>
        <div>
            <h:inputText id="lastName" label="Last Name"
                value="#{userBean.lastName}"></h:inputText>
        </div>
        <div>
            <h:outputLabel for="age">Age</h:outputLabel>
        </div>
        <div>
            <h:inputText id="age" label="age" value="#{userBean.age}">
            </h:inputText>
        </div>
        <div>
            <h:outputLabel for="dateOfBirth">Date of Birth (dd-mm-yyyy)</h:outputLabel>
        </div>
        <div>
            <h:inputText id="dateOfBirth" label="Date of Birth"
                value="#{userBean.dateOfBirth}">
                <f:convertDateTime pattern="dd-mm-yyyy" />
            </h:inputText>
        </div>
        <div>
            <h:outputLabel for="height">Height</h:outputLabel>
        </div>
        <div>
```

```
            <h:inputText id="height" label="Height" value="#{userBean.height}"></h:inputText>
        </div>
        <div> </div>
        <div> </div>
        <div>
            <h:commandButton value="Submit" action="viewuser"></h:commandButton>
        </div>
    </h:form>
</body>
</html>
```

Now modify the

```
viewuser.xhtml
```


attribute of the

```
f:convertNumber
```

tag to achieve this.

```
<?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>View User</title>
</head>
<body>
    <h:form>
        <h2>View User</h2>
        <h4>
            <h:outputText value="#{userBean.firstName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.lastName}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.age}"></h:outputText>
            <br />
            <h:outputText value="#{userBean.dateOfBirth}">
                <f:convertDateTime pattern="dd-MMM-yy"></f:convertDateTime>
            </h:outputText>
            <br />
            <h:outputText value="#{userBean.height}">
                <f:convertNumber maxFractionDigits="1" />
            </h:outputText>
        </h4>
    </h:form>
</body>
</html>
```

Now again package using maven and copy the

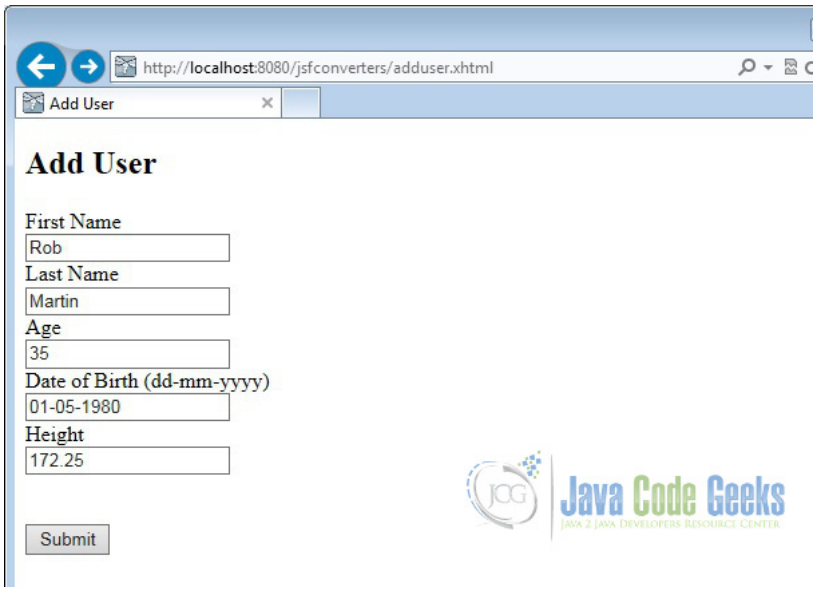
```
war
```

to the apache tomcat

```
webapps
```

folder. Open the following URL in the browser

```
http://localhost:8080/jsfconverters/adduser.xhtml
```

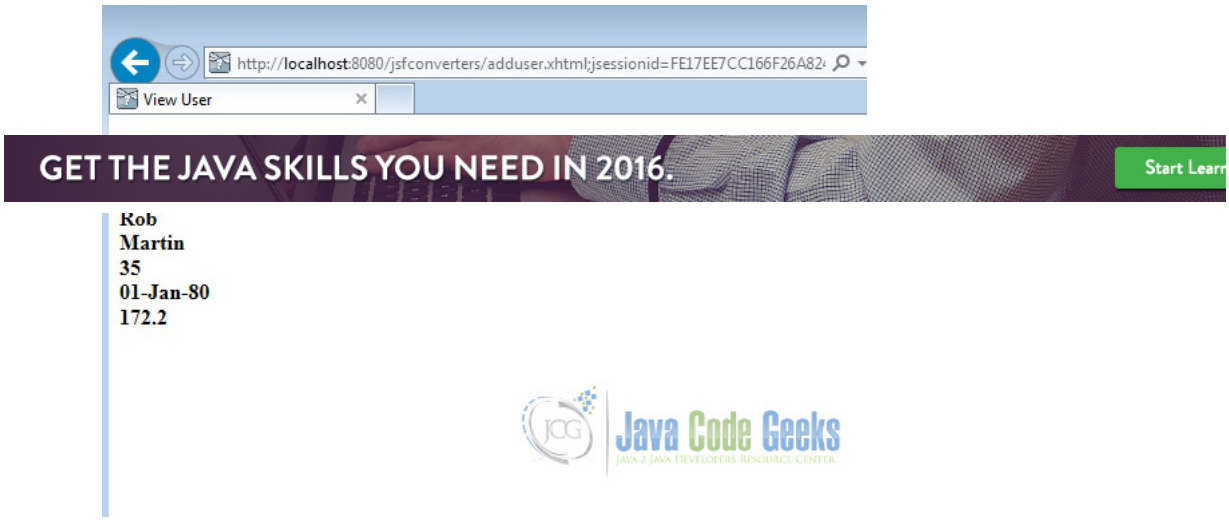


Add User -3

Enter the values for first name, last name, age, date of birth and height (two decimal digits). Now, click on the submit button. The JSF will use the implicit navigation to forward the request to

```
viewuser.xhtml
```

. We will see the height being displayed with single fraction digit irrespective of the user input.



View User -3

9. Download the Eclipse Project

This was an example of how to use Java Server Faces Standard Converters.

Download
You can download the full source code of this example here: **JSF Standard Converters**

Tagged with: JSF CONVERTERS

Courses
News
Resources
Tutorials
Whitepapers

THE CODE GEEKS NETWORK

Java Code Geeks
System Code Geeks
Web Code Geeks

Android Alert Dialog Example
Android OnClickListener Example
How to convert Character to String and a String to Character Array in Java
Java Inheritance example
Java write to File Example
java.io.FileNotFoundException – How to
– 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

JCGs (Java Code Geeks) is an independent online community focused o ultimate Java to Java developers resource center; targeted at the techn technical team lead (senior developer), project manager and junior dev JCGs serve the Java, SOA, Agile and Telecom communities with daily ne domain experts, articles, tutorials, reviews, announcements, code snipp source projects.

DISCLAIMER