About the Tutorial

Java Server Faces (JSF) is a Java-based web application framework intended to simplify development integration of web-based user interfaces. JavaServer Faces is a standardized display technology, which was formalized in a specification through the Java Community Process.

This tutorial will teach you basic JSF concepts and will also take you through various advance concepts related to JSF framework.

Audience

This tutorial has been prepared for the beginners to help them understand basic JSF programming. After completing this tutorial, you will find yourself at a moderate level of expertise in JSF programming from where you can take yourself to the next levels.

Prerequisites

Before proceeding with this tutorial you should have a basic understanding of Java programming language, text editor, and execution of programs etc. Since we are going to develop web-based applications using JSF, it will be good if you have an understanding of other web technologies such as HTML, CSS, AJAX, etc.

Copyright & Disclaimer

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.
# Table of Contents

About the Tutorial........................................................................................................................................... i
Audience .............................................................................................................................................................. i
Prerequisites ....................................................................................................................................................... i
Copyright & Disclaimer................................................................................................................................... i
Table of Contents ............................................................................................................................................... ii

1. JSF – OVERVIEW ......................................................................................................................................... 1
   What is JSF? .................................................................................................................................................... 1
   Benefits ............................................................................................................................................................ 1
   JSF UI Component Model ............................................................................................................................. 1

2. JSF – ENVIRONMENTAL SETUP .............................................................................................................. 2
   System Requirement ...................................................................................................................................... 2
   Environment Setup for JSF Application Development .................................................................................. 2

3. JSF – ARCHITECTURE ............................................................................................................................. 10
   What is MVC Design Pattern? ..................................................................................................................... 10
   JSF Architecture .......................................................................................................................................... 10

4. JSF – LIFE CYCLE ..................................................................................................................................... 12
   Phase 1: Restore view .................................................................................................................................. 12
   Phase 2: Apply request values ..................................................................................................................... 12
   Phase 3: Process validation .......................................................................................................................... 13
   Phase 4: Update model values ..................................................................................................................... 13
   Phase 5: Invoke application .......................................................................................................................... 13
   Phase 6: Render response ............................................................................................................................. 13
9. JSF – FACELET TAGS ................................................................. 191

   Template Tags ........................................................................ 192
   Creating Template .................................................................. 193
   ui:param Tag ......................................................................... 200
   Parameter to Section of a Template ......................................... 201
Parameter to Template ................................................................. 201
Custom Tag .............................................................................. 205
ui:remove Tag ............................................................................ 210

10. JSF - CONVERTOR TAGS .......................................................... 214
    f:convertNumber .................................................................. 214
    f:convertDateTime ................................................................. 218
    Custom Converter ................................................................... 223

11. JSF - VALIDATOR TAGS ............................................................. 231
    f:validateLength .................................................................. 232
    f:validateLongRange ............................................................. 236
    f:validateDoubleRange .......................................................... 239
    f:validateRegex .................................................................... 243
    Custom Validator .................................................................... 247

12. JSF – DATATABLE ................................................................. 254
    Display DataTable .................................................................. 255
    Add Data to DataTable ............................................................ 266
    Edit Data of a DataTable ........................................................ 270
    Delete Data of a DataTable ...................................................... 273
    Using DataModel in a DataTable ............................................ 276

13. JSF – COMPOSITE COMPONENTS ........................................ 280
    Define Custom Component ................................................... 280
    Use Custom Component ........................................................ 282

14. JSF – AJAX ........................................................................... 287
15. JSF – EVENT HANDLING ..............................................................................................................292
    valueChangeListener..................................................................................................................292
    actionListener ..........................................................................................................................300
    Application Events .....................................................................................................................305

16. JSF - JDBC INTEGRATION........................................................................................................312

17. JSF - SPRING INTEGRATION..................................................................................................321

18. JSF - EXPRESSION LANGUAGE................................................................................................331

19. JSF - INTERNATIONALIZATION ..............................................................................................334
What is JSF?

JavaServer Faces (JSF) is a MVC web framework that simplifies the construction of User Interfaces (UI) for server-based applications using reusable UI components in a page. JSF provides a facility to connect UI widgets with data sources and to server-side event handlers. The JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components. JSF enables the reuse and extension of the existing standard UI components.

Benefits

JSF reduces the effort in creating and maintaining applications, which will run on a Java application server and will render application UI on to a target client. JSF facilitates Web application development by -

- Providing reusable UI components
- Making easy data transfer between UI components
- Managing UI state across multiple server requests
- Enabling implementation of custom components
- Wiring client-side event to server-side application code

JSF UI Component Model

JSF provides the developers with the capability to create Web application from collections of UI components that can render themselves in different ways for multiple client types (for example - HTML browser, wireless, or WAP device).

JSF provides -

- Core library
- A set of base UI components - standard HTML input elements
- Extension of the base UI components to create additional UI component libraries or to extend existing components
- Multiple rendering capabilities that enable JSF UI components to render themselves differently depending on the client types
This chapter will guide you on how to prepare a development environment to start your work with JSF Framework. You will learn how to setup JDK, Eclipse, Maven, and Tomcat on your machine before you set up JSF Framework.

**System Requirement**

JSF requires JDK 1.5 or higher so the very first requirement is to have JDK installed on your machine.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JDK</strong></td>
<td>1.5 or above</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>No minimum requirement</td>
</tr>
<tr>
<td><strong>Disk Space</strong></td>
<td>No minimum requirement</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>No minimum requirement</td>
</tr>
</tbody>
</table>

**Environment Setup for JSF Application Development**

Follow the given steps to setup your environment to start with JSF application development.

**Step 1: Verify Java installation on your machine.**

Open console and execute the following **Java** command.

<table>
<thead>
<tr>
<th>OS</th>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows</strong></td>
<td>Open Command Console</td>
<td>c:&gt; java -version</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td>Open Command Terminal</td>
<td>$ java -version</td>
</tr>
<tr>
<td><strong>Mac</strong></td>
<td>Open Terminal</td>
<td>machine:~ joseph$ java -version</td>
</tr>
</tbody>
</table>
Let's verify the output for all the operating systems:

<table>
<thead>
<tr>
<th>OS</th>
<th>Generated Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
<tr>
<td>Linux</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
<tr>
<td>Mac</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
<tr>
<td></td>
<td>Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)</td>
</tr>
</tbody>
</table>

**Step 2: Set Up Java Development Kit (JDK).**

If you do not have Java installed then you can install the Java Software Development Kit (SDK) from Oracle's Java site: [Java SE Downloads](https://www.oracle.com/java/technologies/javase-downloads.html). You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally, set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically `java_install_dir/bin` and `java_install_dir` respectively.

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine.

For example -

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21</td>
</tr>
<tr>
<td>Linux</td>
<td>Export JAVA_HOME=/usr/local/java-current</td>
</tr>
<tr>
<td>Mac</td>
<td>Export JAVA_HOME=/Library/Java/Home</td>
</tr>
</tbody>
</table>
Append Java compiler location to System Path.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Windows</strong></td>
<td>Append the string ;%JAVA_HOME%\bin to the end of the system variable, Path.</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td>Export PATH=$PATH:$JAVA_HOME/bin/</td>
</tr>
<tr>
<td><strong>Mac</strong></td>
<td>Not required</td>
</tr>
</tbody>
</table>

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java. Otherwise, carry out a proper setup according to the given document of the IDE.

**Step 3: Set Up Eclipse IDE.**

All the examples in this tutorial have been written using Eclipse IDE. Hence, we suggest you should have the latest version of Eclipse installed on your machine based on your operating system.

To install Eclipse IDE, download the latest Eclipse binaries with WTP support from [http://www.eclipse.org/downloads/](http://www.eclipse.org/downloads/). Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\eclipse on Windows, or /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.

Eclipse can be started by executing the following commands on Windows machine, or you can simply double-click on eclipse.exe

```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$/usr/local/eclipse/eclipse
```
After a successful startup, if everything is fine then it will display the following result.

*Note: Install m2eclipse plugin to eclipse using the following eclipse software update site

m2eclipse Plugin - http://m2eclipse.sonatype.org/update/

This plugin enables the developers to run maven commands within eclipse with embedded/external maven installation.

**Step 4: Download Maven archive.**

Download Maven 2.2.1 from http://maven.apache.org/download.html

<table>
<thead>
<tr>
<th>OS</th>
<th>Archive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>apache-maven-2.0.11-bin.zip</td>
</tr>
<tr>
<td>Linux</td>
<td>apache-maven-2.0.11-bin.tar.gz</td>
</tr>
<tr>
<td>Mac</td>
<td>apache-maven-2.0.11-bin.tar.gz</td>
</tr>
</tbody>
</table>
Step 5: Extract the Maven archive.

Extract the archive to the directory you wish to install Maven 2.2.1. The subdirectory apache-maven-2.2.1 will be created from the archive.

<table>
<thead>
<tr>
<th>OS</th>
<th>Location (can be different based on your installation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>C:\Program Files\Apache Software Foundation\apache-maven-2.2.1</td>
</tr>
<tr>
<td>Linux</td>
<td>/usr/local/apache-maven</td>
</tr>
<tr>
<td>Mac</td>
<td>/usr/local/apache-maven</td>
</tr>
</tbody>
</table>

Step 6: Set Maven environment variables.

Add M2_HOME, M2, MAVEN_OPTS to environment variables.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variables using system properties.</td>
</tr>
<tr>
<td></td>
<td>M2_HOME=C:\Program Files\Apache Software Foundation\apache-maven-2.2.1</td>
</tr>
<tr>
<td></td>
<td>M2=%M2_HOME%\bin</td>
</tr>
<tr>
<td></td>
<td>MAVEN_OPTS=-Xms256m -Xmx512m</td>
</tr>
<tr>
<td>Linux</td>
<td>Open command terminal and set environment variables.</td>
</tr>
<tr>
<td></td>
<td>export M2_HOME=/usr/local/apache-maven/apache-maven-2.2.1</td>
</tr>
<tr>
<td></td>
<td>export M2=%M2_HOME%\bin</td>
</tr>
<tr>
<td></td>
<td>export MAVEN_OPTS=-Xms256m -Xmx512m</td>
</tr>
<tr>
<td>Mac</td>
<td>Open command terminal and set environment variables.</td>
</tr>
<tr>
<td></td>
<td>export M2_HOME=/usr/local/apache-maven/apache-maven-2.2.1</td>
</tr>
<tr>
<td></td>
<td>export M2=%M2_HOME%\bin</td>
</tr>
<tr>
<td></td>
<td>export MAVEN_OPTS=-Xms256m -Xmx512m</td>
</tr>
</tbody>
</table>
Step 7: Add Maven bin directory location to system path.

Now append M2 variable to System Path.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Append the string ;%M2% to the end of the system variable, Path.</td>
</tr>
<tr>
<td>Linux</td>
<td>export PATH=$M2:$PATH</td>
</tr>
<tr>
<td>Mac</td>
<td>export PATH=$M2:$PATH</td>
</tr>
</tbody>
</table>

Step 8: Verify Maven installation.

Open console, execute the following mvn command.

<table>
<thead>
<tr>
<th>OS</th>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Open Command Console</td>
<td>c:&gt; mvn --version</td>
</tr>
<tr>
<td>Linux</td>
<td>Open Command Terminal</td>
<td>$ mvn --version</td>
</tr>
<tr>
<td>Mac</td>
<td>Open Terminal</td>
<td>machine:~ joseph$ mvn --version</td>
</tr>
</tbody>
</table>

Finally, verify the output of the above commands, which should be as shown in the following table.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)</td>
</tr>
<tr>
<td></td>
<td>Java version: 1.6.0_21</td>
</tr>
<tr>
<td></td>
<td>Java home: C:\Program Files\Java\jdk1.6.0_21\jre</td>
</tr>
<tr>
<td>Linux</td>
<td>Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)</td>
</tr>
<tr>
<td></td>
<td>Java version: 1.6.0_21</td>
</tr>
<tr>
<td></td>
<td>Java home: C:\Program Files\Java\jdk1.6.0_21\jre</td>
</tr>
<tr>
<td>Mac</td>
<td>Apache Maven 2.2.1 (r801777; 2009-08-07 00:46:01+0530)</td>
</tr>
<tr>
<td></td>
<td>Java version: 1.6.0_21</td>
</tr>
<tr>
<td></td>
<td>Java home: C:\Program Files\Java\jdk1.6.0_21\jre</td>
</tr>
</tbody>
</table>
Step 9: Set Up Apache Tomcat.

You can download the latest version of Tomcat from http://tomcat.apache.org/. Once you download the installation, unpack the binary distribution into a convenient location. For example, in C:\apache-tomcat-6.0.33 on Windows, or /usr/local/apache-tomcat-6.0.33 on Linux/Unix and set CATALINA_HOME environment variable pointing to the installation locations.

Tomcat can be started by executing the following commands on Windows machine, or you can simply double-click on startup.bat

```bash
%CATALINA_HOME%\bin\startup.bat
```

or

```bash
C:\apache-tomcat-6.0.33\bin\startup.bat
```

Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```bash
$CATALINA_HOME/bin/startup.sh
```

or

```bash
/usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting http://localhost:8080/. If everything is fine, then it will display the following result.
Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: http://tomcat.apache.org

Tomcat can be stopped by executing the following commands on Windows machine.

```bash
%CATALINA_HOME%\bin\shutdown
```

or

```bash
C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine.

```bash
$CATALINA_HOME/bin/shutdown.sh
```

or

```bash
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```
JSF technology is a framework for developing, building server-side User Interface Components and using them in a web application. JSF technology is based on the Model View Controller (MVC) architecture for separating logic from presentation.

**What is MVC Design Pattern?**

MVC design pattern designs an application using three separate modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Carries Data and login</td>
</tr>
<tr>
<td>View</td>
<td>Shows User Interface</td>
</tr>
<tr>
<td>Controller</td>
<td>Handles processing of an application</td>
</tr>
</tbody>
</table>

The purpose of MVC design pattern is to separate model and presentation enabling developers to focus on their core skills and collaborate more clearly.

Web designers have to concentrate only on view layer rather than model and controller layer. Developers can change the code for model and typically need not change view layer. Controllers are used to process user actions. In this process, layer model and views may be changed.

**JSF Architecture**

JSF application is similar to any other Java technology-based web application; it runs in a Java servlet container, and contains -

- JavaBeans components as models containing application-specific functionality and data
- A custom tag library for representing event handlers and validators
- A custom tag library for rendering UI components
- UI components represented as stateful objects on the server
- Server-side helper classes
- Validators, event handlers, and navigation handlers
- Application configuration resource file for configuring application resources

There are controllers which can be used to perform user actions. UI can be created by web page authors and the business logic can be utilized by managed beans.

JSF provides several mechanisms for rendering an individual component. It is upto the web page designer to pick the desired representation, and the application developer doesn't need to know which mechanism was used to render a JSF UI component.
JSF application life cycle consists of six phases which are as follows -

- Restore view phase
- Apply request values phase; process events
- Process validations phase; process events
- Update model values phase; process events
- Invoke application phase; process events
- Render response phase

The six phases show the order in which JSF processes a form. The list shows the phases in their likely order of execution with event processing at each phase.

**Phase 1: Restore view**

JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request.

During this phase, JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance. The FacesContext instance will now contain all the information required to process a request.

**Phase 2: Apply request values**

After the component tree is created/restored, each component in the component tree uses the decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.

If any decode methods event listeners called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.
Phase 3: Process validation

During this phase, JSF processes all validators registered on the component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component.

If the local value is invalid, JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and displays the same page again with the error message.

Phase 4: Update model values

After the JSF checks that the data is valid, it walks over the component tree and sets the corresponding server-side object properties to the components' local values. JSF will update the bean properties corresponding to the input component's value attribute.

If any updateModels methods called renderResponse on the current FacesContext instance, JSF moves to the render response phase.

Phase 5: Invoke application

During this phase, JSF handles any application-level events, such as submitting a form/linking to another page.

Phase 6: Render response

During this phase, JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as JSP container executes the page. If this is not an initial request, the component tree is already built so components need not be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page.

After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.
5. JSF – First Application

To create a simple JSF application, we’ll use maven-archetype-webapp plugin. In the following example, we’ll create a maven-based web application project in C:\JSF folder.

Create Project

Let’s open command console, go the C:\>JSF directory and execute the following **mvn** command.

```
C:\JSF>mvn archetype:create
-DgroupId=com.tutorialspoint.test
-DartifactId=helloworld
-DarchetypeArtifactId=maven-archetype-webapp
```

Maven will start processing and will create the complete java web application project structure.

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:create] (aggregator-style)
[INFO] ------------------------------------------------------------------------
[INFO] [archetype:create {execution: default-cli}]
[INFO] Defaulting package to group ID: com.tutorialspoint.test
[INFO] artifact org.apache.maven.archetypes:maven-archetype-webapp:
checking for updates from central
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project
from Old (1.x) Archetype: maven-archetype-webapp:RELEASE
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.tutorialspoint.test
[INFO] Parameter: packageName, Value: com.tutorialspoint.test
[INFO] Parameter: package, Value: com.tutorialspoint.test
[INFO] Parameter: artifactId, Value: helloworld
[INFO] Parameter: basedir, Value: C:\JSF
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir:
```
Now go to C:/JSF directory. You'll see a Java web application project created, named helloworld (as specified in artifactId). Maven uses a standard directory layout as shown in the following screenshot.

Using the above example, we can understand the following key concepts.

<table>
<thead>
<tr>
<th>Folder Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>helloworld</td>
<td>Contains src folder and pom.xml</td>
</tr>
<tr>
<td>src/main/wepapp</td>
<td>Contains WEB-INF folder and index.jsp page</td>
</tr>
<tr>
<td>src/main/resources</td>
<td>It contains images/properties files (In the above example, we need to create this structure manually)</td>
</tr>
</tbody>
</table>

**Add JSF Capability to Project**

Add the following JSF dependencies.

```xml
<dependencies>
  <dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.1.7</version>
  </dependency>
</dependencies>
```
<dependency>
    <groupId>com.sun.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>2.1.7</version>
</dependency>

</dependencies>

Complete 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.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld 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>2.1.7</version>
    </dependency>
  </dependencies>

</project>
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.1.7</version>
</dependency>

</dependencies>

<build>
  <finalName>helloworld</finalName>

  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>

      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>

    </plugin>

  </plugins>

</build>

</project>

---

**Prepare Eclipse Project**

Let's open the command console. Go the C:\ > JSF > helloworld directory and execute the following **mvn** command.

```
C:\JSF\helloworld>mvn eclipse:eclipse -Dwtpversion=2.0
```

Maven will start processing, create the eclipse ready project, and will add wtp capability.
[INFO] Searching repository for plugin with prefix: 'eclipse'.
[INFO] -------------------------------------------------------------
[INFO] Building helloworld Maven Webapp
[INFO]  task-segment: [eclipse:eclipse]
[INFO] -------------------------------------------------------------
[INFO] Preparing eclipse:eclipse
[INFO] No goals needed for project - skipping
[INFO] [eclipse:eclipse {execution: default-cli}]
[INFO] Adding support for WTP version 2.0.
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Total time: 6 minutes 7 seconds
Import Project in Eclipse

Following are the steps:

- Import project in eclipse using Import wizard.
- Go to **File -> Import... -> Existing project into workspace.**
- Select root directory to helloworld.
- Keep **Copy projects into workspace** to be checked.
- Click Finish button.
- Eclipse will import and copy the project in its workspace **C: \ -> Projects \ -> Data \ -> WorkSpace.**

Configure Faces Servlet in web.xml

Locate web.xml in **webapp \ -> WEB-INF** folder and update it as shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
http="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">

<welcome-file-list>
  <welcome-file>faces/home.xhtml</welcome-file>
</welcome-file-list>
```
FacesServlet is main servlet responsible to handle all request. It acts as central controller. This servlet initializes the JSF components before the JSP is displayed. -->

<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>/faces/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>

</web-app>
Create a Managed Bean

Create a package structure under `src -> main -> java as com -> tutorialspoint -> test`. Create HelloWorld.java class in this package. Update the code of HelloWorld.java as shown below.

```java
package com.tutorialspoint.test;

import javax.faces.bean.ManagedBean;

@ManagedBean(name = "helloWorld", eager = true)
public class HelloWorld {
    public HelloWorld() {
        System.out.println("HelloWorld started!");
    }

    public String getMessage() {
        return "Hello World!";
    }
}
```

Create a JSF page

Create a page home.xhtml under webapp folder. Update the code of home.xhtml as shown below.

```html
<!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">
    <head>
        <title>JSF Tutorial!</title>
    </head>

    <body>
        #{helloWorld.getMessage()}
    </body>
</html>
```
**Build the Project**

Following are the steps.

- Select helloworld project in eclipse
- Use Run As wizard
- Select **Run As -> Maven package**
- Maven will start building the project and will create helloworld.war under C:\ Projects -> Data -> WorkSpace -> helloworld -> target folder.

```
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------
[INFO] Building helloworld Maven Webapp
[INFO] [INFO] Id: com.tutorialspoint.test:helloworld:war:1.0-SNAPSHOT
[INFO] task-segment: [package]
[INFO] ------------------------------------------------------
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] No sources to compile
[INFO] [surefire:test]
[INFO] Surefire report directory: C:\Projects\Data\WorkSpace\helloworld\target\surefire-reports

-------------------------------------------------------
    TESTS
-------------------------------------------------------

There are no tests to run.

Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO] [war:war]
[INFO] Packaging webapp
[INFO] Assembling webapp[helloworld] in
```
[INFO] Processing war project
[INFO] Webapp assembled in [150 msecs]
[INFO] Building war:
C:\Projects\Data\WorkSpace\helloworld\target\helloworld
[INFO] -----------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] -----------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Mon Nov 05 16:34:46 IST 2012
[INFO] Final Memory: 2M/15M
[INFO] -----------------------------------------------

**Deploy WAR file**

Following are the steps.

- Stop the tomcat server.
- Copy the helloworld.war file to **tomcat installation directory -> webapps folder**.
- Start the tomcat server.
- Look inside webapps directory, there should be a folder **helloworld** got created.
- Now helloworld.war is successfully deployed in Tomcat Webserver root.

**Run Application**

Enter a url in web browser: **http://localhost:8080/helloworld/home.jsf** to launch the application.

Server name (localhost) and port (8080) may vary as per your tomcat configuration.
Managed Bean is a regular Java Bean class registered with JSF. In other words, Managed Beans is a Java bean managed by JSF framework. Managed bean contains the getter and setter methods, business logic, or even a backing bean (a bean contains all the HTML form value).

Managed beans works as Model for UI component. Managed Bean can be accessed from JSF page.

In **JSF 1.2**, a managed bean had to register it in JSF configuration file such as faces-config.xml. From **JSF 2.0** onwards, managed beans can be easily registered using annotations. This approach keeps beans and its registration at one place hence it becomes easier to manage.

**Using XML Configuration**

```xml
<managed-bean>
    <managed-bean-name>helloWorld</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.HelloWorld</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

<managed-bean>
    <managed-bean-name>message</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.Message</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

**Using Annotation**

```java
@ManagedBean(name = "helloWorld", eager = true)
@RequestScoped
public class HelloWorld {

    @ManagedProperty(value="#{message}"
    private Message message;

    ...
}
```
@ManagedBean Annotation

@ManagedBean marks a bean to be a managed bean with the name specified in name attribute. If the name attribute is not specified, then the managed bean name will default to class name portion of the fully qualified class name. In our case, it would be helloWorld.

Another important attribute is eager. If eager="true" then managed bean is created before it is requested for the first time otherwise "lazy" initialization is used in which bean will be created only when it is requested.

Scope Annotations

Scope annotations set the scope into which the managed bean will be placed. If the scope is not specified, then bean will default to request scope. Each scope is briefly discussed in the following table.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@RequestScoped</td>
<td>Bean lives as long as the HTTP request-response lives. It gets created upon a HTTP request and gets destroyed when the HTTP response associated with the HTTP request is finished.</td>
</tr>
<tr>
<td>@NoneScoped</td>
<td>Bean lives as long as a single EL evaluation. It gets created upon an EL evaluation and gets destroyed immediately after the EL evaluation.</td>
</tr>
<tr>
<td>@ViewScoped</td>
<td>Bean lives as long as the user is interacting with the same JSF view in the browser window/tab. It gets created upon a HTTP request and gets destroyed once the user postbacks to a different view.</td>
</tr>
<tr>
<td>@SessionScoped</td>
<td>Bean lives as long as the HTTP session lives. It gets created upon the first HTTP request involving this bean in the session and gets destroyed when the HTTP session is invalidated.</td>
</tr>
<tr>
<td>@ApplicationScoped</td>
<td>Bean lives as long as the web application lives. It gets created upon the first HTTP request involving this bean in the application (or when the web application starts up and the eager=true attribute is set in @ManagedBean) and gets destroyed when the web application shuts down.</td>
</tr>
<tr>
<td>@CustomScoped</td>
<td>Bean lives as long as the bean's entry in the custom Map, which is created for this scope lives.</td>
</tr>
</tbody>
</table>

@ManagedProperty Annotation

JSF is a simple static Dependency Injection (DI) framework. Using @ManagedProperty annotation, a managed bean's property can be injected in another managed bean.
Example Application

Let us create a test JSF application to test the above annotations for managed beans.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - Create Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>HelloWorld.java</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>Message.java</em> under a package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>5</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>6</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**HelloWorld.java**

```java
package com.tutorialspoint.test;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.RequestScoped;

@ManagedBean(name = "helloworld", eager = true)
@RequestScoped
public class HelloWorld {

    @ManagedProperty(value="#{message}"")
    private Message messageBean;
    private String message;
    public HelloWorld() {
        System.out.println("HelloWorld started!");
    }
    public String getMessage() {
        if(messageBean != null){
            message = messageBean.getMessage();
        }
```
return message;
}
public void setMessageBean(Message message) {
    this.messageBean = message;
}
}

Message.java

package com.tutorialspoint.test;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean(name = "message", eager = true)
@RequestScoped
public class Message {
    private String message = "Hello World!";
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    }

home.xhtml

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    #{helloWorld.message}
</body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.
Navigation rules are those rules provided by JSF Framework that describes which view is to be shown when a button or a link is clicked.

Navigation rules can be defined in JSF configuration file named faces-config.xml. They can be defined in managed beans.

Navigation rules can contain conditions based on which the resulted view can be shown. JSF 2.0 provides implicit navigation as well in which there is no need to define navigation rules as such.

**Implicit Navigation**

JSF 2.0 provides **auto view page resolver** mechanism named **implicit navigation**. In this case, you only need to put view name in **action** attribute and JSF will search the correct **view** page automatically in the deployed application.

**Auto Navigation in JSF Page**

Set view name in action attribute of any JSF UI Component.

```xml
<h:form>
    <h3>Using JSF outcome</h3>
    <h:commandButton action="page2" value="Page2" />
</h:form>
```
Here, when **Page2** button is clicked, JSF will resolve the view name, **page2** as page2.xhtml extension, and find the corresponding view file **page2.xhtml** in the current directory.

### Auto Navigation in Managed Bean

Define a method in managed bean to return a view name.

```java
@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
class NavigationController implements Serializable {
    public String moveToPage1() {
        return "page1";
    }
}
```

Get view name in action attribute of any JSF UI Component using managed bean.

```html
<h:form>
    <h3>Using Managed Bean</h3>
    <h:commandButton action="#{navigationController.moveToPage1}"
    value="Page1" />
</h:form>
```

Here, when **Page1** button is clicked, JSF will resolve the view name, **page1** as page1.xhtml extension, and find the corresponding view file **page1.xhtml** in the current directory.
Conditional Navigation

Using managed bean, we can very easily control the navigation. Look at the following code in a managed bean.

```java
@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
public class NavigationController implements Serializable {

    // this managed property will read value from request parameter pageId
    @ManagedProperty(value="#{param.pageId}")
    private String pageId;

    // conditional navigation based on pageId
    // if pageId is 1 show page1.xhtml,
```
//if pageId is 2 show page2.xhtml
//else show home.xhtml
public String showPage(){
    if(pageId == null){
        return "home";
    }
    if(pageId.equals("1")){
        return "page1";
    }else if(pageId.equals("2")){
        return "page2";
    }else{
        return "home";
    }
}
}

Pass pageId as a request parameter in JSF UI Component.

```
<h:form>
    <h:commandLink action="#{navigationController.showPage}" value="Page1">
        <f:param name="pageId" value="1" />
    </h:commandLink>
</h:form>

<%-- other buttons --%>
```

Here, when "Page1" button is clicked.

- JSF will create a request with parameter pageId=1
- Then JSF will pass this parameter to managed property pageId of navigationController
- Now navigationController.showPage() is called which will return view as page1 after checking the pageId
- JSF will resolve the view name, page1 as page1.xhtml extension
- Find the corresponding view file page1.xhtml in the current directory

Resolving Navigation Based on `from-action`

JSF provides navigation resolution option even if managed bean different methods returns the same view name.

Look at the following code in a managed bean.

```java
public String processPage1()
{
    return "page";
}
public String processPage2()
{
    return "page";
}
```
To resolve views, define the following navigation rules in `faces-config.xml`

```xml
<navigation-rule>
  <from-view-id>home.xhtml</from-view-id>
  <navigation-case>
    <from-action>${navigationController.processPage1}</from-action>
    <from-outcome>page</from-outcome>
    <to-view-id>page1.jsf</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>${navigationController.processPage2}</from-action>
    <from-outcome>page</from-outcome>
    <to-view-id>page2.jsf</to-view-id>
  </navigation-case>
</navigation-rule>
```

Here, when **Page1** button is clicked -

- **navigationController.processPage1()** is called which will return view as page
- JSF will resolve the view name, **page1** as view name is **page and from-action** in `faces-config` is **navigationController.processPage1**
- Find the corresponding view file **page1.xhtml** in the current directory

---

**Forward vs Redirect**

JSF by default performs a server page forward while navigating to another page and the URL of the application does not change.

To enable the page redirection, append **faces-redirect=true** at the end of the view name.
前进与重定向导航

前进

Page1

重定向

Page1

<h:form>
  <h3>前进</h3>
  <h:commandButton action="page1" value="Page1" />
  <h3>重定向</h3>
  <h:commandButton action="page1?faces-redirect=true" value="Page1" />
</h:form>

在这里，当Page1按钮下的前进被点击时，你将得到以下结果。
Here when **Page1** button under **Redirect** is clicked, you will get the following result.

![Example Application](image)

**Example Application**

Let us create a test JSF application to test all of the above navigation examples.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - Create Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create <em>NavigationController.java</em> under a package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>faces-config.xml</em> under a <em>WEB-INF</em> folder and updated its contents as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Update <em>web.xml</em> under a <em>WEB-INF</em> folder as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Create <em>page1.xhtml</em> and <em>page2.xhtml</em> and modify <em>home.xhtml</em> under a <em>webapp</em> folder as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
NavigationController.java

```java
package com.tutorialspoint.test;

import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ManagedProperty;
import javax.faces.bean.RequestScoped;

@ManagedBean(name = "navigationController", eager = true)
@RequestScoped
public class NavigationController implements Serializable {

    private static final long serialVersionUID = 1L;

    @ManagedProperty(value="#{param.pageId}")
    private String pageId;

    public String moveToPage1(){
        return "page1";
    }

    public String moveToPage2(){
        return "page2";
    }

    public String moveToHomePage(){
        return "home";
    }

    public String processPage1(){
        return "page";
    }

    public String processPage2(){
        return "page";
    }
}
```
public String showPage()
{
    if (pageId == null)
    {
        return "home";
    }
    if (pageId.equals("1"))
    {
        return "page1";
    } else if (pageId.equals("2"))
    {
        return "page2";
    } else{
        return "home";
    }
}

public String getPageId() {
    return pageId;
}

public void setPageId(String pageId) {
    this.pageId = pageId;
}

faces-config.xml

<?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/javae/web-facesconfig_2_0.xsd"
    version="2.0">
    <navigation-rule>
        <from-view-id>home.xhtml</from-view-id>
        <navigation-case>
            <from-action>${navigationController.processPage1}</from-action>
            <from-outcome>page</from-outcome>
            <to-view-id>page1.jsf</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-action>${navigationController.processPage2}</from-action>
        </navigation-case>
    </navigation-rule>
</faces-config>
<from-outcome>page</from-outcome>
<to-view-id>page2.jsf</to-view-id>
</navigation-case>
</navigation-rule>
</navigation-case>
</faces-config>

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>

<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-name>
  <param-value>Development</param-value>
</context-param>
<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
</web-app>
page1.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:body>
    <h2>This is Page1</h2>
    <h:form>
      <h:commandButton action="home?faces-redirect=true"
                        value="Back To Home Page" />
    </h:form>
  </h:body>
</html>
```

page2.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:body>
    <h2>This is Page2</h2>
    <h:form>
      <h:commandButton action="home?faces-redirect=true"
                        value="Back To Home Page" />
    </h:form>
  </h:body>
</html>
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">

<body>
  <h2>Implicit Navigation</h2>
  <hr />
  <form>
    <h3>Using Managed Bean</h3>
    <h:commandButton action="#{navigationController.moveToPage1}"
                      value="Page1" />
    <h3>Using JSF outcome</h3>
    <h:commandButton action="page2" value="Page2" />
  </form>
  <br />
  <h2>Conditional Navigation</h2>
  <hr />
  <form>
    <h:commandLink action="#{navigationController.showPage}"
                    value="Page1">
      <f:param name="pageId" value="1" />
    </h:commandLink>

    <h:commandLink action="#{navigationController.showPage}"
                    value="Page2">
      <f:param name="pageId" value="2" />
    </h:commandLink>

    <h:commandLink action="#{navigationController.showPage}"
                    value="Home">
      <f:param name="pageId" value="3" />
    </h:commandLink>
  </form>
  <br />
</body>
```

41
<h2>"From Action" Navigation</h2>
<hr />
<h:form>
    <h:commandLink action="#{navigationController.processPage1}"
        value="Page1" />

    <h:commandLink action="#{navigationController.processPage2}"
        value="Page2" />
</h:form>
<br />
<h2>Forward vs Redirection Navigation</h2>
<hr />
<h:form>
    <h:commandButton action="page1" value="Page1" />
    <h3>Forward</h3>
    <h:commandButton action="page1?faces-redirect=true"
        value="Page1" />
    <h3>Redirect</h3>
</h:form>
</h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.
In this chapter, you will learn about various types of basic JSF tags.

JSF provides a standard HTML tag library. These tags get rendered into corresponding html output.

For these tags you need to use the following namespaces of URI in html node.

```html
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
>
```

Following are the important *Basic Tags* in JSF 2.0.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>h:inputText</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a HTML input of type=&quot;text&quot;, text box.</td>
</tr>
<tr>
<td>2</td>
<td><strong>h:inputSecret</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a HTML input of type=&quot;password&quot;, text box.</td>
</tr>
<tr>
<td>3</td>
<td><strong>h:inputTextarea</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a HTML textarea field.</td>
</tr>
<tr>
<td>4</td>
<td><strong>h:inputHidden</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a HTML input of type=&quot;hidden&quot;.</td>
</tr>
<tr>
<td>5</td>
<td><strong>h:selectBooleanCheckbox</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a single HTML check box.</td>
</tr>
<tr>
<td>6</td>
<td><strong>h:selectManyCheckbox</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a group of HTML check boxes.</td>
</tr>
<tr>
<td>7</td>
<td><strong>h:selectOneRadio</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a single HTML radio button.</td>
</tr>
<tr>
<td>8</td>
<td><strong>h:selectOneListbox</strong></td>
</tr>
<tr>
<td></td>
<td>Renders a HTML single list box.</td>
</tr>
<tr>
<td>No.</td>
<td>Tag</td>
</tr>
<tr>
<td>-----</td>
<td>------------------</td>
</tr>
<tr>
<td>9</td>
<td><code>h:selectManyListbox</code></td>
</tr>
<tr>
<td>10</td>
<td><code>h:selectOneMenu</code></td>
</tr>
<tr>
<td>11</td>
<td><code>h:outputText</code></td>
</tr>
<tr>
<td>12</td>
<td><code>h:outputFormat</code></td>
</tr>
<tr>
<td>13</td>
<td><code>h:graphicImage</code></td>
</tr>
<tr>
<td>14</td>
<td><code>h:outputStylesheet</code></td>
</tr>
<tr>
<td>15</td>
<td><code>h:outputScript</code></td>
</tr>
<tr>
<td>16</td>
<td><code>h:commandButton</code></td>
</tr>
<tr>
<td>17</td>
<td><code>h:Link</code></td>
</tr>
<tr>
<td>18</td>
<td><code>h:commandLink</code></td>
</tr>
<tr>
<td>19</td>
<td><code>h:outputLink</code></td>
</tr>
<tr>
<td>20</td>
<td><code>h:panelGrid</code></td>
</tr>
<tr>
<td>21</td>
<td><code>h:message</code></td>
</tr>
</tbody>
</table>
### h:messages
Renders all messages for JSF UI Components.

### f:param
Pass parameters to JSF UI Component.

### f:attribute
Pass attribute to a JSF UI Component.

### f:setPropertyActionListener
Sets value of a managed bean's property.

---

### h:inputText
The h:inputText tag renders an HTML input element of the type "text".

#### JSF Tag
```xml
<h:inputText value="Hello World!" />
```

#### Rendered Output
```html
<input type="text" name="j_idt6:j_idt8" value="Hello World!" />
```

#### Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
</tbody>
</table>

- **id**: Identifier for a component
- **binding**: Reference to the component that can be used in a backing bean
- **rendered**: A boolean; false suppresses rendering
- **styleClass**: Cascading stylesheet (CSS) class name
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><strong>value</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
<td></td>
</tr>
</tbody>
</table>

| 6 | **valueChangeListener**  |
|   | A method binding to a method that responds to value changes |

| 7 | **converter**  |
|   | Converter class name |

| 8 | **validator**  |
|   | Class name of a validator that’s created and attached to a component |

| 9 | **required**  |
|   | A boolean; if true, requires a value to be entered in the associated field |

| 10 | **accesskey**  |
|   | A key, typically combined with a system-defined metakey, that gives focus to an element |

| 11 | **accept**  |
|   | Comma-separated list of content types for a form |

| 12 | **accept-charset**  |
|   | Comma- or space-separated list of character encodings for a form. The **accept-charset** attribute is specified with the JSF HTML attribute named **acceptcharset**. |

| 13 | **alt**  |
|   | Alternative text for nontextual elements such as images or applets |

| 14 | **border**  |
|   | Pixel value for an element’s border width |

| 15 | **charset**  |
|   | Character encoding for a linked resource |

<p>| 16 | <strong>coords</strong>  |
|   | Coordinates for an element whose shape is a rectangle, circle, or polygon |</p>
<table>
<thead>
<tr>
<th>17</th>
<th><strong>dir</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direction for text. Valid values are <strong>ltr</strong> (left to right) and <strong>rtl</strong> (right to left).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>18</th>
<th><strong>disabled</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Disabled state of an input element or button</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19</th>
<th><strong>hreflang</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base language of a resource specified with the <strong>href</strong> attribute; <strong>hreflang</strong> may only be used with <strong>href</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>20</th>
<th><strong>lang</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base language of an element’s attributes and text</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>21</th>
<th><strong>maxlength</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maximum number of characters for text fields</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>22</th>
<th><strong>readonly</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read-only state of an input field; the text can be selected in a readonly field but not edited</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23</th>
<th><strong>style</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inline style information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>24</th>
<th><strong>tabindex</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numerical value specifying a tab index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25</th>
<th><strong>target</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The name of a frame in which a document is opened</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>26</th>
<th><strong>title</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>27</th>
<th><strong>type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type of a link; for example, <strong>stylesheet</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>28</strong></td>
<td><strong>width</strong>&lt;br&gt;Width of an element</td>
</tr>
<tr>
<td><strong>29</strong></td>
<td><strong>onblur</strong>&lt;br&gt;Element loses focus</td>
</tr>
<tr>
<td><strong>30</strong></td>
<td><strong>onchange</strong>&lt;br&gt;Element’s value changes</td>
</tr>
<tr>
<td><strong>31</strong></td>
<td><strong>onclick</strong>&lt;br&gt;Mouse button is clicked over the element</td>
</tr>
<tr>
<td><strong>32</strong></td>
<td><strong>ondblclick</strong>&lt;br&gt;Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td><strong>33</strong></td>
<td><strong>onfocus</strong>&lt;br&gt;Element receives focus</td>
</tr>
<tr>
<td><strong>34</strong></td>
<td><strong>onkeydown</strong>&lt;br&gt;Key is pressed</td>
</tr>
<tr>
<td><strong>35</strong></td>
<td><strong>onkeypress</strong>&lt;br&gt;Key is pressed and subsequently released</td>
</tr>
<tr>
<td><strong>36</strong></td>
<td><strong>onkeyup</strong>&lt;br&gt;Key is released</td>
</tr>
<tr>
<td><strong>37</strong></td>
<td><strong>onmousedown</strong>&lt;br&gt;Mouse button is pressed over the element</td>
</tr>
<tr>
<td><strong>38</strong></td>
<td><strong>onmousemove</strong>&lt;br&gt;Mouse moves over the element</td>
</tr>
<tr>
<td><strong>39</strong></td>
<td><strong>onmouseout</strong>&lt;br&gt;Mouse leaves the element’s area</td>
</tr>
</tbody>
</table>
### Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
home.xhtml

```html
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:inputText example</h2>
    <hr />
    <h:form>
        <h:inputText value="Hello World!" readonly="true"/>
        <h3>Read-Only input text box</h3>
        <h:inputText value="Hello World"/>
        <h3>Normal input text box</h3>
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![Image of the result](http://localhost:8080/helloworld/home.xhtml)
**h:inputSecret**

The h:inputSecret tag renders an HTML input element of the type "password".

**JSF Tag**

```xml
<h:inputSecret value="password"/>
```

**Rendered Output**

```xml
<input type="password" name="j_idt12:j_idt16" value="password"/>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td>9</td>
<td>required</td>
</tr>
</tbody>
</table>

- **id**: Identifier for a component
- **binding**: Reference to the component that can be used in a backing bean
- **rendered**: A boolean; false suppresses rendering
- **styleClass**: Cascading stylesheet (CSS) class name
- **value**: A component’s value, typically a value binding
- **valueChangeListener**: A method binding to a method that responds to value changes
- **converter**: Converter class name
- **validator**: Class name of a validator that’s created and attached to a component
- **required**: A boolean; if true, requires a value to be entered in the associated field
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accesskey</strong></td>
<td>A key, typically combined with a system-defined meta-key, that gives focus to an element</td>
</tr>
<tr>
<td><strong>accept</strong></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td><strong>accept-charset</strong></td>
<td>Comma- or space-separated list of character encodings for a form. The <strong>accept-charset</strong> attribute is specified with the JSF HTML attribute named <strong>acceptcharset</strong></td>
</tr>
<tr>
<td><strong>alt</strong></td>
<td>Alternative text for nontextual elements such as images or applets</td>
</tr>
<tr>
<td><strong>border</strong></td>
<td>Pixel value for an element’s border width</td>
</tr>
<tr>
<td><strong>charset</strong></td>
<td>Character encoding for a linked resource</td>
</tr>
<tr>
<td><strong>coords</strong></td>
<td>Coordinates for an element whose shape is a rectangle, circle, or polygon</td>
</tr>
<tr>
<td><strong>dir</strong></td>
<td>Direction for text. Valid values are <strong>ltr</strong> (left to right) and <strong>rtl</strong> (right to left).</td>
</tr>
<tr>
<td><strong>disabled</strong></td>
<td>Disabled state of an input element or button</td>
</tr>
<tr>
<td><strong>hreflang</strong></td>
<td>Base language of a resource specified with the <strong>href</strong> attribute; <strong>hreflang</strong> may only be used with <strong>href</strong></td>
</tr>
<tr>
<td><strong>lang</strong></td>
<td>Base language of an element’s attributes and text</td>
</tr>
<tr>
<td><strong>maxlength</strong></td>
<td>Maximum number of characters for text fields</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 22 | **readonly**  
Read-only state of an input field; text can be selected in a readonly field but not edited |
| 23 | **style**  
Inline style information |
| 24 | **tabindex**  
Numerical value specifying a tab index |
| 25 | **target**  
The name of a frame in which a document is opened |
| 26 | **title**  
A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value |
| 27 | **type**  
Type of a link; for example, *stylesheet* |
| 28 | **width**  
Width of an element |
| 29 | **onblur**  
Element loses focus |
| 30 | **onchange**  
Element’s value changes |
| 31 | **onclick**  
Mouse button is clicked over the element |
| 32 | **ondblclick**  
Mouse button is double-clicked over the element |
| 33 | **onfocus**  
Element receives focus |
| 34 | **onkeydown**  
Key is pressed |
<table>
<thead>
<tr>
<th></th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td><code>onkeypress</code></td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td>36</td>
<td><code>onkeyup</code></td>
<td>Key is released</td>
</tr>
<tr>
<td>37</td>
<td><code>onmousedown</code></td>
<td>Mouse button is pressed over the element</td>
</tr>
<tr>
<td>38</td>
<td><code>onmousemove</code></td>
<td>Mouse moves over the element</td>
</tr>
<tr>
<td>39</td>
<td><code>onmouseout</code></td>
<td>Mouse leaves the element's area</td>
</tr>
<tr>
<td>40</td>
<td><code>onmouseover</code></td>
<td>Mouse moves onto an element</td>
</tr>
<tr>
<td>41</td>
<td><code>onmouseup</code></td>
<td>Mouse button is released</td>
</tr>
<tr>
<td>42</td>
<td><code>onreset</code></td>
<td>Form is reset</td>
</tr>
<tr>
<td>43</td>
<td><code>onselect</code></td>
<td>Text is selected in an input field</td>
</tr>
<tr>
<td>44</td>
<td><code>immediate</code></td>
<td>Process validation early in the life cycle</td>
</tr>
<tr>
<td>45</td>
<td><code>redisplay</code></td>
<td>when true, the input field’s value is redisplayed when the web page is reloaded</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
</head>
<body>
  <h2>h:inputSecret example</h2>
  <hr />
  <h:form>
    <h:inputSecret value="password" readonly="true"/>
    <h3>Read-Only input password box</h3>
  </h:form>
</body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

### h:inputTextarea

The h:inputTextarea tag renders an HTML input element of the type "text".

**JSF Tag**

```html
<h:inputTextarea row="10" col="10" value="Hello World! Everything is fine!" readonly="true"/>
```

**Rendered Output**

```html
<textarea name="j_idt18:j_idt20" readonly="readonly">
Hello World! Everything is fine!
</textarea>
```

**Tag Attributes**
<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td>9</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td>10</td>
<td>accesskey</td>
</tr>
<tr>
<td></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td>11</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td>12</td>
<td>accept-charset</td>
</tr>
</tbody>
</table>
Comma- or space-separated list of character encodings for a form. The **acceptcharset** attribute is specified with the JSF HTML attribute named **acceptcharset**.

<table>
<thead>
<tr>
<th>Column</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td><strong>cols</strong></td>
<td>Number of columns</td>
</tr>
<tr>
<td>14</td>
<td><strong>border</strong></td>
<td>Pixel value for an element’s border width</td>
</tr>
<tr>
<td>15</td>
<td><strong>charset</strong></td>
<td>Character encoding for a linked resource</td>
</tr>
<tr>
<td>16</td>
<td><strong>coords</strong></td>
<td>Coordinates for an element whose shape is a rectangle, circle, or polygon</td>
</tr>
<tr>
<td>17</td>
<td><strong>dir</strong></td>
<td>Direction for text. Valid values are ltr (left to right) and rtl (right to left)</td>
</tr>
<tr>
<td>18</td>
<td><strong>disabled</strong></td>
<td>Disabled state of an input element or button</td>
</tr>
<tr>
<td>19</td>
<td><strong>hreflang</strong></td>
<td>Base language of a resource specified with the <strong>href</strong> attribute; <strong>hreflang</strong> may only be used with <strong>href</strong>.</td>
</tr>
<tr>
<td>20</td>
<td><strong>lang</strong></td>
<td>Base language of an element’s attributes and text</td>
</tr>
<tr>
<td>21</td>
<td><strong>rows</strong></td>
<td>Number of rows</td>
</tr>
<tr>
<td>22</td>
<td><strong>readonly</strong></td>
<td>Read-only state of an input field; the text can be selected in a readonly field but not edited</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>23</td>
<td><strong>style</strong></td>
<td>Inline style information</td>
</tr>
<tr>
<td>24</td>
<td><strong>tabindex</strong></td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>25</td>
<td><strong>target</strong></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>26</td>
<td><strong>title</strong></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>27</td>
<td><strong>type</strong></td>
<td>Type of a link; for example, <em>stylesheet</em></td>
</tr>
<tr>
<td>28</td>
<td><strong>width</strong></td>
<td>Width of an element</td>
</tr>
<tr>
<td>29</td>
<td><strong>onblur</strong></td>
<td>Element loses focus</td>
</tr>
<tr>
<td>30</td>
<td><strong>onchange</strong></td>
<td>Element’s value changes</td>
</tr>
<tr>
<td>31</td>
<td><strong>onclick</strong></td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td>32</td>
<td><strong>ondblclick</strong></td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>33</td>
<td><strong>onfocus</strong></td>
<td>Element receives focus</td>
</tr>
<tr>
<td>34</td>
<td><strong>onkeydown</strong></td>
<td>Key is pressed</td>
</tr>
</tbody>
</table>
onkeypress  
Key is pressed and subsequently released

onkeyup  
Key is released

onmousedown  
Mouse button is pressed over the element

onmousemove  
Mouse moves over the element

onmouseout  
Mouse leaves the element’s area

onmouseover  
Mouse moves onto an element

onmouseup  
Mouse button is released

onreset  
Form is reset

onselect  
Text is selected in an input field

immediate  
Process validation early in the life cycle

Example Application
Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```html
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>h:inputTextArea example</h2>
    <hr />
    <h:form>
        <h3>Read-Only input text area</h3>
        <h:inputTextarea row="10" col="10" value="Hello World! Everything is fine!" readonly="true" />
        <br />
        <h3>Normal input text area</h3>
        <h:inputTextarea value="Hello World! Everything is fine!" />
    </h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
h:inputHidden

The h:inputHidden tag renders an HTML input element of the type "hidden".

JSF Tag

```xml
<h:inputHidden value="Hello World" id="hiddenField"/>
```

Rendered Output

```html
<input id="jsfForm:hiddenField" type="hidden" name="jsfForm:hiddenField" value="Hello World"/>
```

Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>Property</td>
</tr>
<tr>
<td>---</td>
<td>-----------------</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td>9</td>
<td>required</td>
</tr>
<tr>
<td>10</td>
<td>accesskey</td>
</tr>
<tr>
<td>11</td>
<td>accept</td>
</tr>
<tr>
<td>12</td>
<td>accept-charset</td>
</tr>
<tr>
<td>13</td>
<td>cols</td>
</tr>
<tr>
<td>14</td>
<td>border</td>
</tr>
<tr>
<td>15</td>
<td>charset</td>
</tr>
<tr>
<td>16</td>
<td>coords</td>
</tr>
<tr>
<td>17</td>
<td>dir</td>
</tr>
<tr>
<td>18</td>
<td>disabled</td>
</tr>
<tr>
<td>19</td>
<td>hreflang</td>
</tr>
<tr>
<td>20</td>
<td>lang</td>
</tr>
<tr>
<td>21</td>
<td>rows</td>
</tr>
<tr>
<td>22</td>
<td>readonly</td>
</tr>
<tr>
<td>23</td>
<td>style</td>
</tr>
<tr>
<td>24</td>
<td>tabindex</td>
</tr>
<tr>
<td>25</td>
<td>target</td>
</tr>
<tr>
<td>26</td>
<td>title</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>27</td>
<td><strong>type</strong>&lt;br&gt;Type of a link; for example, stylesheet</td>
</tr>
<tr>
<td>28</td>
<td><strong>width</strong>&lt;br&gt;Width of an element</td>
</tr>
<tr>
<td>29</td>
<td><strong>onblur</strong>&lt;br&gt;Element loses focus</td>
</tr>
<tr>
<td>30</td>
<td><strong>onchange</strong>&lt;br&gt;Element's value changes</td>
</tr>
<tr>
<td>31</td>
<td><strong>onclick</strong>&lt;br&gt;Mouse button is clicked over the element</td>
</tr>
<tr>
<td>32</td>
<td><strong>ondblclick</strong>&lt;br&gt;Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>33</td>
<td><strong>onfocus</strong>&lt;br&gt;Element receives focus</td>
</tr>
<tr>
<td>34</td>
<td><strong>onkeydown</strong>&lt;br&gt;Key is pressed</td>
</tr>
<tr>
<td>35</td>
<td><strong>onkeypress</strong>&lt;br&gt;Key is pressed and subsequently released</td>
</tr>
<tr>
<td>36</td>
<td><strong>onkeyup</strong>&lt;br&gt;Key is released</td>
</tr>
<tr>
<td>37</td>
<td><strong>onmousedown</strong>&lt;br&gt;Mouse button is pressed over the element</td>
</tr>
<tr>
<td>38</td>
<td><strong>onmousemove</strong>&lt;br&gt;Mouse moves over the element</td>
</tr>
</tbody>
</table>

A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value.
Example Application
Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the <code>JSF - First Application</code> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
</tbody>
</table>
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```xml
<!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">
<head>
<title>JSF Tutorial!</title>
<h:head>
<script type="text/javascript">
  function showHiddenValue(){
    alert(document.getElementById('jsfForm:hiddenField').value);
  }
</script>
</h:head>
</head>
<body>
<h2>h:inputHidden example</h2>
<hr />
<h:form id="jsfForm">
<h3>Get value from inputHidden field</h3>
<h:inputHidden value="Hello World" id="hiddenField" />
<h:commandButton value="Show Hidden Value" onclick="showHiddenValue()" />
</h:form>
</body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.
The `h:selectBooleanCheckbox` tag renders an HTML input element of the type "checkbox".

**JSF Tag**

```xml
<h:selectBooleanCheckbox value="Remember Me" id="chkRememberMe" />
```

**Rendered Output**

```html
<input id="jsfForm1:chkRememberMe" type="checkbox" name="jsfForm1:chkRememberMe" checked="checked" />
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
</tbody>
</table>

Identifier for a component
<table>
<thead>
<tr>
<th></th>
<th><strong>binding</strong></th>
<th>Reference to the component that can be used in a backing bean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>rendered</strong></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td></td>
<td><strong>styleClass</strong></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td></td>
<td><strong>value</strong></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td></td>
<td><strong>valueChangeListener</strong></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td></td>
<td><strong>converter</strong></td>
<td>Converter class name</td>
</tr>
<tr>
<td></td>
<td><strong>validator</strong></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td></td>
<td><strong>accesskey</strong></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td></td>
<td><strong>accept</strong></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>12</td>
<td>accept-charslet</td>
<td>Comma- or space-separated list of character encodings for a form. The accept-charslet attribute is specified with the JSF HTML attribute named acceptcharset.</td>
</tr>
<tr>
<td>13</td>
<td>alt</td>
<td>Alternative text for non-textual elements such as images or applets.</td>
</tr>
<tr>
<td>14</td>
<td>charset</td>
<td>Character encoding for a linked resource.</td>
</tr>
<tr>
<td>15</td>
<td>coords</td>
<td>Coordinates for an element whose shape is a rectangle, circle, or polygon.</td>
</tr>
<tr>
<td>16</td>
<td>dir</td>
<td>Direction for text. Valid values are ltr (left to right) and rtl (right to left).</td>
</tr>
<tr>
<td>17</td>
<td>disabled</td>
<td>Disabled state of an input element or button.</td>
</tr>
<tr>
<td>18</td>
<td>hreflang</td>
<td>Base language of a resource specified with the href attribute; hreflang may only be used with href.</td>
</tr>
<tr>
<td>19</td>
<td>lang</td>
<td>Base language of an element’s attributes and text.</td>
</tr>
<tr>
<td>20</td>
<td>maxlength</td>
<td>Maximum number of characters for text fields.</td>
</tr>
<tr>
<td>21</td>
<td>readonly</td>
<td>Read-only state of an input field; text can be selected in a readonly field but not edited.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>22</td>
<td><strong>rel</strong></td>
<td>Relationship between the current document and a link specified with the <strong>href</strong> attribute</td>
</tr>
<tr>
<td>23</td>
<td><strong>rev</strong></td>
<td>Reverse link from the anchor specified with <strong>href</strong> to the current document. The value of the attribute is a space-separated list of link types.</td>
</tr>
<tr>
<td>24</td>
<td><strong>rows</strong></td>
<td>Number of visible rows in a text area. <strong>h:dataTable</strong> has a <strong>rows</strong> attribute, but it’s not an HTML pass-through attribute.</td>
</tr>
<tr>
<td>25</td>
<td><strong>shape</strong></td>
<td>Shape of a region. Valid values: <strong>default, rect, circle, poly</strong>. (default signifies the entire region)</td>
</tr>
<tr>
<td>26</td>
<td><strong>style</strong></td>
<td>Inline style information</td>
</tr>
<tr>
<td>27</td>
<td><strong>tabindex</strong></td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>28</td>
<td><strong>target</strong></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>29</td>
<td><strong>title</strong></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>30</td>
<td><strong>type</strong></td>
<td>Type of a link; for example, <strong>stylesheet</strong></td>
</tr>
<tr>
<td>31</td>
<td><strong>width</strong></td>
<td>Width of an element</td>
</tr>
<tr>
<td></td>
<td>Event Name</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>32</td>
<td><code>onblur</code></td>
<td>Element loses focus</td>
</tr>
<tr>
<td>33</td>
<td><code>onchange</code></td>
<td>Element’s value changes</td>
</tr>
<tr>
<td>34</td>
<td><code>onclick</code></td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td>35</td>
<td><code>ondbliclick</code></td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>36</td>
<td><code>onfocus</code></td>
<td>Element receives focus</td>
</tr>
<tr>
<td>37</td>
<td><code>onkeydown</code></td>
<td>Key is pressed</td>
</tr>
<tr>
<td>38</td>
<td><code>onkeypress</code></td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td>39</td>
<td><code>onkeyup</code></td>
<td>Key is released</td>
</tr>
<tr>
<td>40</td>
<td><code>onmousedown</code></td>
<td>Mouse button is pressed over the element</td>
</tr>
<tr>
<td>41</td>
<td><code>onmousemove</code></td>
<td>Mouse moves over the element</td>
</tr>
<tr>
<td>42</td>
<td><code>onmouseout</code></td>
<td>Mouse leaves the element’s area</td>
</tr>
<tr>
<td>43</td>
<td><code>onmouseover</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>onmouseup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse button is released</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>onreset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Form is reset</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>onselect</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Text is selected in an input field</td>
<td></td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
</tbody>
</table>
Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
  <h:head>
    <script type="text/javascript">
      function showCheckedValue(){
        alert(document.getElementById('jsfForm1:chkRememberMe').checked);
      }
    </script>
  </h:head>
</head>

<h2>h:selectBooleanCheckbox example</h2>
<hr />
<form id="jsfForm1">
  <h3>Get value from selectBooleanCheckbox field</h3>
  <h:selectBooleanCheckbox value="Remember Me" id="chkRememberMe" />
  <h:commandButton value="Show Checked" onclick="showCheckedValue()" />
</form>
</body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

### h:selectBooleanCheckbox example

Get value from selectBooleanCheckbox field

![Image of checkbox example]

### h:selectManyCheckbox

The h:selectManyCheckbox tag renders a set of HTML input element of type "checkbox", and format it with HTML table and label tags.

#### JSF Tag

```
<h:selectManyCheckbox value="#{userData.data}"
    >
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectManyCheckbox>
```

#### Rendered Output

```
<table>
    <tr>
        <td><input name="j_idt6:j_idt8" id="j_idt6:j_idt8:0" value="1"
                    type="checkbox" checked="checked" />
            <label for="j_idt6:j_idt8:0" class=""> Item 1</label>
        </td>
        <td><input name="j_idt6:j_idt8" id="j_idt6:j_idt8:1" value="2"
                    type="checkbox" checked="checked" />
            <label for="j_idt6:j_idt8:1" class=""> Item 2</label>
        </td>
    </tr>
</table>
```
Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component's value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
</tr>
<tr>
<td>9</td>
<td><strong>required</strong>&lt;br&gt;A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td>10</td>
<td><strong>accesskey</strong>&lt;br&gt;A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td>11</td>
<td><strong>accept</strong>&lt;br&gt;Comma-separated list of content types for a form</td>
</tr>
<tr>
<td>12</td>
<td><strong>accept_charset</strong>&lt;br&gt;Comma- or space-separated list of character encodings for a form. The <code>accept_charset</code> attribute is specified with the JSF HTML attribute named <code>acceptcharset</code></td>
</tr>
<tr>
<td>13</td>
<td><strong>alt</strong>&lt;br&gt;Alternative text for nontextual elements such as images or applets</td>
</tr>
<tr>
<td>14</td>
<td><strong>charset</strong>&lt;br&gt;Character encoding for a linked resource</td>
</tr>
<tr>
<td>15</td>
<td><strong>coords</strong>&lt;br&gt;Coordinates for an element whose shape is a rectangle, circle, or polygon</td>
</tr>
<tr>
<td>16</td>
<td><strong>dir</strong>&lt;br&gt;Direction for text. Valid values are <code>ltr</code> (left to right) and <code>rtl</code> (right to left)</td>
</tr>
<tr>
<td>17</td>
<td><strong>disabled</strong>&lt;br&gt;Disabled state of an input element or button</td>
</tr>
<tr>
<td>18</td>
<td><strong>hreflang</strong>&lt;br&gt;Base language of a resource specified with the <code>href</code> attribute; <code>hreflang</code> may only be used with <code>href</code></td>
</tr>
</tbody>
</table>
| 19 | **lang**  
Base language of an element’s attributes and text |
| 20 | **maxlength**  
Maximum number of characters for text fields |
| 21 | **readonly**  
Read-only state of an input field; text can be selected in a readonly field but not edited |
| 22 | **rel**  
Relationship between the current document and a link specified with the *href* attribute |
| 23 | **rev**  
Reverse link from the anchor specified with *href* to the current document. The value of the attribute is a space-separated list of link types |
| 24 | **rows**  
Number of visible rows in a text area. *h: dataTable* has a **rows** attribute, but it’s not an HTML pass-through attribute |
| 25 | **shape**  
Shape of a region. Valid values: *default*, *rect*, *circle*, *poly*. (default signifies the entire region) |
| 26 | **style**  
Inline style information |
| 27 | **tabindex**  
Numerical value specifying a tab index |
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td><strong>target</strong></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>29</td>
<td><strong>title</strong></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>30</td>
<td><strong>type</strong></td>
<td>Type of a link; for example, <em>stylesheet</em></td>
</tr>
<tr>
<td>31</td>
<td><strong>width</strong></td>
<td>Width of an element</td>
</tr>
<tr>
<td>32</td>
<td><strong>onblur</strong></td>
<td>Element loses focus</td>
</tr>
<tr>
<td>33</td>
<td><strong>onchange</strong></td>
<td>Element’s value changes</td>
</tr>
<tr>
<td>34</td>
<td><strong>onclick</strong></td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td>35</td>
<td><strong>ondblclick</strong></td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>36</td>
<td><strong>onfocus</strong></td>
<td>Element receives focus</td>
</tr>
<tr>
<td>37</td>
<td><strong>onkeydown</strong></td>
<td>Key is pressed</td>
</tr>
<tr>
<td>38</td>
<td><strong>onkeypress</strong></td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td></td>
<td>Event Description</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>onkeyup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Key is released</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>onmousedown</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse button is pressed over the element</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>onmousemove</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse moves over the element</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>onmouseout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse leaves the element’s area</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>onmouseover</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse moves onto an element</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>onmouseup</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mouse button is released</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>onreset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Form is reset</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>onselect</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Text is selected in an input field</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>disabledClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSS class for disabled elements</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>enabledClass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSS class for enabled elements</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>layout</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specification for how elements are laid out: lineDirection (horizontal) or pageDirection (vertical)</td>
<td></td>
</tr>
</tbody>
</table>
**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify home.xhtml as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create result.xhtml in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create UserData.java as a managed bean under package com.tutorialspoint.test as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String[] data = {"1","2","3");

    public String[] getData() {
        return data;
    }

    public void setData(String[] data) {
        this.data = data;
    }
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html">
<head>
    <title>JSF Tutorial!</title>
</head>
<h:body>
    <h2>h:selectManyCheckbox example</h2>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.
Select multiple checkboxes and press **Submit** button. We've selected four items. You will see the selected results.

**h:selectManyCheckbox example**

Mutiple checkboxes

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

Submit

**h:selectOneRadio**

The **h:selectOneRadio** tag renders a set of HTML input element of type "radio". Format it with HTML table and label tags.

**JSF Tag**

```xml
<h:selectOneRadio value="#{userData.data}"
    >
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectOneRadio>
```
Rendered Output

```html
<table>
  <tr>
    <td><input type="radio" checked="checked" name="j_idt6:j_idt8"
      id="j_idt6:j_idt8:0" value="1" />
      <label for="j_idt6:j_idt8:0"> Item 1</label></td>
    <td><input type="radio" name="j_idt6:j_idt8"
      id="j_idt6:j_idt8:1" value="2" />
      <label for="j_idt6:j_idt8:1"> Item 2</label></td>
  </tr>
</table>
```

Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 8 | **validator**  
Class name of a validator that's created and attached to a component |
| 9 | **required**  
A boolean; if true, requires a value to be entered in the associated field |
| 10 | **accesskey**  
A key, typically combined with a system-defined metakey, that gives focus to an element |
| 11 | **accept**  
Comma-separated list of content types for a form |
| 12 | **accept-charset**  
Comma- or space-separated list of character encodings for a form. The `accept-charset` attribute is specified with the JSF HTML attribute named `acceptcharset` |
| 13 | **alt**  
Alternative text for nontextual elements such as images or applets |
| 14 | **charset**  
Character encoding for a linked resource |
| 15 | **coords**  
Coordinates for an element whose shape is a rectangle, circle, or polygon |
| 16 | **dir**  
Direction for text. Valid values are ltr (left to right) and rtl (right to left) |
| 17 | **disabled**  
Disabled state of an input element or button |
<p>| 18 | <strong>hreflang</strong> |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base language of a resource specified with the href attribute; hreflang may only be used with href</strong></td>
<td></td>
</tr>
<tr>
<td><strong>lang</strong></td>
<td>Base language of an element’s attributes and text</td>
</tr>
<tr>
<td><strong>maxlength</strong></td>
<td>Maximum number of characters for text fields</td>
</tr>
<tr>
<td><strong>readonly</strong></td>
<td>Read-only state of an input field; text can be selected in a readonly field but not edited</td>
</tr>
<tr>
<td><strong>rel</strong></td>
<td>Relationship between the current document and a link specified with the href attribute</td>
</tr>
<tr>
<td><strong>rev</strong></td>
<td>Reverse link from the anchor specified with href to the current document. The value of the attribute is a space-separated list of link types</td>
</tr>
<tr>
<td><strong>rows</strong></td>
<td>Number of visible rows in a text area. h: dataTable has a rows attribute, but it’s not an HTML pass-through attribute</td>
</tr>
<tr>
<td><strong>shape</strong></td>
<td>Shape of a region. Valid values: default, rect, circle, poly. (default signifies the entire region)</td>
</tr>
<tr>
<td><strong>style</strong></td>
<td>Inline style information</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 27 | **tabindex**  
Numerical value specifying a tab index |
| 28 | **target**  
The name of a frame in which a document is opened |
| 29 | **title**  
A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value |
| 30 | **type**  
Type of a link; for example, *stylesheet* |
| 31 | **width**  
Width of an element |
| 32 | **onblur**  
Element loses focus |
| 33 | **onchange**  
Element’s value changes |
| 34 | **onclick**  
Mouse button is clicked over the element |
| 35 | **ondblclick**  
Mouse button is double-clicked over the element |
| 36 | **onfocus**  
Element receives focus |
| 37 | **onkeydown**  
Key is pressed |
| 38 | **onkeypress**  
Key is pressed and subsequently released |
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td><strong>onkeyup</strong>&lt;br&gt;Key is released</td>
</tr>
<tr>
<td>40</td>
<td><strong>onmousedown</strong>&lt;br&gt;Mouse button is pressed over the element</td>
</tr>
<tr>
<td>41</td>
<td><strong>onmousemove</strong>&lt;br&gt;Mouse moves over the element</td>
</tr>
<tr>
<td>42</td>
<td><strong>onmouseout</strong>&lt;br&gt;Mouse leaves the element’s area</td>
</tr>
<tr>
<td>43</td>
<td><strong>onmouseover</strong>&lt;br&gt;Mouse moves onto an element</td>
</tr>
<tr>
<td>44</td>
<td><strong>onmouseup</strong>&lt;br&gt;Mouse button is released</td>
</tr>
<tr>
<td>45</td>
<td><strong>onreset</strong>&lt;br&gt;Form is reset</td>
</tr>
<tr>
<td>46</td>
<td><strong>onselect</strong>&lt;br&gt;Text is selected in an input field</td>
</tr>
<tr>
<td>47</td>
<td><strong>disabledClass</strong>&lt;br&gt;CSS class for disabled elements</td>
</tr>
<tr>
<td>48</td>
<td><strong>enabledClass</strong>&lt;br&gt;CSS class for enabled elements</td>
</tr>
<tr>
<td>49</td>
<td><strong>layout</strong>&lt;br&gt;Specification for how elements are laid out: lineDirection (horizontal) or pageDirection (vertical)</td>
</tr>
<tr>
<td>50</td>
<td><strong>border</strong>&lt;br&gt;Border of the element</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <em>UserData.java</em> as a managed bean under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
UserData.java

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Select any option and press **Submit** button. We’ve selected item 3. You will see the selected results.

**h:selectManyCheckbox example**

Radio Button

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

Submit

**Result**

3

**h:selectOneListbox**

The **h:selectOneListbox** tag renders an HTML input element of the type "select" with size specified.

**JSF Tag**

```xml
<h:selectOneListbox value="#{userData.data}">
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
</h:selectOneListbox>
```
</h:selectOneListbox>

**Rendered Output**

```xml
<select name="j_idt6:j_idt8" size="2">
   <option value="1">Item 1</option>
   <option value="2">Item 2</option>
</select>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>required</strong></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td><strong>accesskey</strong></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td><strong>accept</strong></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td><strong>accept-charset</strong></td>
<td>Comma- or space-separated list of character encodings for a form. The <strong>accept-charset</strong> attribute is specified with the <strong>JSF HTML attribute named acceptcharset</strong></td>
</tr>
<tr>
<td><strong>alt</strong></td>
<td>Alternative text for nontextual elements such as images or applets</td>
</tr>
<tr>
<td><strong>charset</strong></td>
<td>Character encoding for a linked resource</td>
</tr>
<tr>
<td><strong>coords</strong></td>
<td>Coordinates for an element whose shape is a rectangle, circle, or polygon</td>
</tr>
<tr>
<td><strong>dir</strong></td>
<td>Direction for text. Valid values are <strong>ltr</strong> (left to right) and <strong>rtl</strong> (right to left)</td>
</tr>
<tr>
<td><strong>disabled</strong></td>
<td>Disabled state of an input element or button</td>
</tr>
<tr>
<td><strong>hreflang</strong></td>
<td>Base language of a resource specified with the <strong>href</strong> attribute; <strong>hreflang</strong> may only be used with <strong>href</strong></td>
</tr>
<tr>
<td><strong>lang</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>20</td>
<td><strong>base language</strong> of an element’s attributes and text</td>
</tr>
</tbody>
</table>
| 21 | **maxlength**  
Maximum number of characters for text fields |
| 21 | **readonly**  
Read-only state of an input field; text can be selected in a readonly field but not edited |
| 22 | **rel**  
Relationship between the current document and a link specified with the `href` attribute |
| 23 | **rev**  
Reverse link from the anchor specified with `href` to the current document. The value of the attribute is a space-separated list of link types |
| 24 | **rows**  
Number of visible rows in a text area. `h: dataTable` has a `rows` attribute, but it’s not an HTML pass-through attribute |
| 25 | **shape**  
Shape of a region. Valid values: `default, rect, circle, poly` (default signifies the entire region) |
| 26 | **style**  
Inline style information |
| 27 | **tabindex**  
Numerical value specifying a tab index |
<p>| 28 | <strong>target</strong> |</p>
<table>
<thead>
<tr>
<th>29</th>
<th>The name of a frame in which a document is opened</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>30</td>
<td>Type of a link; for example, <strong>stylesheet</strong></td>
</tr>
<tr>
<td>31</td>
<td>Width of an element</td>
</tr>
<tr>
<td>32</td>
<td>Element loses focus</td>
</tr>
<tr>
<td>33</td>
<td>Element’s value changes</td>
</tr>
<tr>
<td>34</td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td>35</td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>36</td>
<td>Element receives focus</td>
</tr>
<tr>
<td>37</td>
<td>Key is pressed</td>
</tr>
<tr>
<td>38</td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td>39</td>
<td>Key is released</td>
</tr>
<tr>
<td></td>
<td>Event</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
</tr>
<tr>
<td>40</td>
<td>onmousedown</td>
</tr>
<tr>
<td>41</td>
<td>onmousemove</td>
</tr>
<tr>
<td>42</td>
<td>onmouseout</td>
</tr>
<tr>
<td>43</td>
<td>onmouseover</td>
</tr>
<tr>
<td>44</td>
<td>onmouseup</td>
</tr>
<tr>
<td>45</td>
<td>onreset</td>
</tr>
<tr>
<td>46</td>
<td>onselect</td>
</tr>
<tr>
<td>47</td>
<td>size</td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>result.xhtml</code> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>UserData.java</code> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";
```
public String getData() {
    return data;
}

public void setData(String data) {
    this.data = data;
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <h:body>
        <h2>h::selectOneListbox example</h2>
        <hr />
        <h:form>
            <h:selectOneListbox value="#{userData.data}"
                f:selectItem itemValue="1" itemLabel="Item 1" />
            <f:selectItem itemValue="2" itemLabel="Item 2" />
            <f:selectItem itemValue="3" itemLabel="Item 3" />
            <f:selectItem itemValue="4" itemLabel="Item 4" />
            <f:selectItem itemValue="5" itemLabel="Item 5" />
        </h:selectOneListbox>
        <h:commandButton value="Submit" action="result" />
    </h:form>
</h:body>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Select any option and press **Submit** button. We’ve selected item 3. You will see the selected results.

```html
<h:selectManyListbox value="#{userData.data}"
   
   <f:selectItem itemValue="1" itemLabel="Item 1" />
   <f:selectItem itemValue="2" itemLabel="Item 2" />

</h:selectManyListbox>
```

**h:selectManyListbox**

The `h:selectManyListbox` tag renders an HTML input element of the type "select" with **size** and **multiple** specified.

**JSF Tag**

```html
<select name="j_idt6:j_idt8" size="2" multiple="multiple">
   <option value="1">Item 1</option>
   <option value="2">Item 2</option>
</select>
```

**Rendered Output**
## Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td>9</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td>10</td>
<td>accesskey</td>
</tr>
<tr>
<td></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td></td>
<td>Property</td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
</tr>
<tr>
<td>11</td>
<td><code>accept</code></td>
</tr>
<tr>
<td>12</td>
<td><code>accept-charset</code></td>
</tr>
<tr>
<td>13</td>
<td><code>alt</code></td>
</tr>
<tr>
<td>14</td>
<td><code>charset</code></td>
</tr>
<tr>
<td>15</td>
<td><code>coords</code></td>
</tr>
<tr>
<td>16</td>
<td><code>dir</code></td>
</tr>
<tr>
<td>17</td>
<td><code>disabled</code></td>
</tr>
<tr>
<td>18</td>
<td><code>hreflang</code></td>
</tr>
<tr>
<td>19</td>
<td><code>lang</code></td>
</tr>
<tr>
<td>20</td>
<td><code>maxlength</code></td>
</tr>
<tr>
<td>21</td>
<td><code>readonly</code></td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>rel</td>
<td>Relationship between the current document and a link specified with the <code>href</code> attribute</td>
</tr>
<tr>
<td>rev</td>
<td>Reverse link from the anchor specified with <code>href</code> to the current document. The value of the attribute is a space-separated list of link types</td>
</tr>
<tr>
<td>rows</td>
<td>Number of visible rows in a text area. <code>h:dataTable</code> has a <code>rows</code> attribute, but it's not an HTML pass-through attribute</td>
</tr>
<tr>
<td>shape</td>
<td>Shape of a region. Valid values: <code>default</code>, <code>rect</code>, <code>circle</code>, <code>poly</code>. (default signifies the entire region)</td>
</tr>
<tr>
<td>style</td>
<td>Inline style information</td>
</tr>
<tr>
<td>tabindex</td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>target</td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>title</td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>type</td>
<td>Type of a link; for example, <code>stylesheet</code></td>
</tr>
<tr>
<td></td>
<td>Event</td>
</tr>
<tr>
<td>---</td>
<td>-----------</td>
</tr>
<tr>
<td>31</td>
<td>width</td>
</tr>
<tr>
<td>32</td>
<td>onblur</td>
</tr>
<tr>
<td>33</td>
<td>onchange</td>
</tr>
<tr>
<td>34</td>
<td>onclick</td>
</tr>
<tr>
<td>35</td>
<td>ondblclick</td>
</tr>
<tr>
<td>36</td>
<td>onfocus</td>
</tr>
<tr>
<td>37</td>
<td>onkeydown</td>
</tr>
<tr>
<td>38</td>
<td>onkeypress</td>
</tr>
<tr>
<td>39</td>
<td>onkeyup</td>
</tr>
<tr>
<td>40</td>
<td>onmousedown</td>
</tr>
<tr>
<td>41</td>
<td>onmousemove</td>
</tr>
<tr>
<td>42</td>
<td>onmouseout</td>
</tr>
<tr>
<td>43</td>
<td>onmouseover</td>
</tr>
</tbody>
</table>
Mouse moves onto an element

<table>
<thead>
<tr>
<th>44</th>
<th>onmouseup</th>
<th>Mouse button is released</th>
</tr>
</thead>
</table>

Form is reset

<table>
<thead>
<tr>
<th>45</th>
<th>onreset</th>
<th>Form is reset</th>
</tr>
</thead>
</table>

Text is selected in an input field

<table>
<thead>
<tr>
<th>46</th>
<th>onselect</th>
<th>Text is selected in an input field</th>
</tr>
</thead>
</table>

Size of input field

<table>
<thead>
<tr>
<th>47</th>
<th>size</th>
<th>Size of input field</th>
</tr>
</thead>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Create <em>UserData.java</em> as a managed bean under package <em>com.tutorialspoint.test</em> as explained below.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
<td></td>
</tr>
</tbody>
</table>
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String[] data = {"1","2","3"};

    public String[] getData() {
        return data;
    }

    public void setData(String[] data) {
        this.data = data;
    }
}
```

**home.xhtml**
<?xml version="1.0" encoding="UTF-8"?>
<!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:fx="http://java.sun.com/jsf/core"
  <h:body>
    <h:selectManyListbox example</h2>
    <hr />
    <h:form>
      <h3>List Box</h3>
      <h:selectManyListbox value="#{userData.data}"
        itemValue="1" itemLabel="Item 1" />
      <f:selectItem itemValue="2" itemLabel="Item 2" />
      <f:selectItem itemValue="3" itemLabel="Item 3" />
      <f:selectItem itemValue="4" itemLabel="Item 4" />
      <f:selectItem itemValue="5" itemLabel="Item 5" />
    </h:selectManyListbox>
    <h:commandButton value="Submit" action="result" />
  </h:form>
</h:body>
</html>

result.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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:fx="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <h2>Result</h2>
  </h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.

Select multiple values and press Submit button. We've selected four items. You will see the selected results.
**h:selectOneMenu**

The `h:selectOneMenu` tag renders an HTML input element of the type "select" with `size` not specified.

**JSF Tag**

```xml
<h:selectOneMenu value="#{userData.data}"
    
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />

</h:selectOneMenu>
```

**Rendered Output**

```xml
<select name="j_idt6:j_idt8">
    <option value="1">Item 1</option>
    <option value="2">Item 2</option>
</select>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>id</code></td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td><code>binding</code></td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td><code>rendered</code></td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td><code>styleClass</code></td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td></td>
<td><strong>value</strong></td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td></td>
<td><strong>valueChangeListener</strong></td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td></td>
<td><strong>converter</strong></td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td></td>
<td><strong>validator</strong></td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td></td>
<td><strong>required</strong></td>
</tr>
<tr>
<td></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td></td>
<td><strong>accesskey</strong></td>
</tr>
<tr>
<td></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td></td>
<td><strong>accept</strong></td>
</tr>
<tr>
<td></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td></td>
<td><strong>accept-charset</strong></td>
</tr>
<tr>
<td></td>
<td>Comma- or space-separated list of character encodings for a form. The <code>accept-charset</code> attribute is specified with the JSF HTML attribute named <code>acceptcharset</code></td>
</tr>
<tr>
<td></td>
<td><strong>alt</strong></td>
</tr>
<tr>
<td></td>
<td>Alternative text for nontextual elements such as images or applets</td>
</tr>
<tr>
<td></td>
<td><strong>charset</strong></td>
</tr>
<tr>
<td></td>
<td>Character encoding for a linked resource</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 15 | **coords**  
Coordinates for an element whose shape is a rectangle, circle, or polygon |
| 16 | **dir**  
Direction for text. Valid values are **ltr** (left to right) and **rtl** (right to left) |
| 17 | **disabled**  
Disabled state of an input element or button |
| 18 | **hreflang**  
Base language of a resource specified with the **href** attribute; **hreflang** may only be used with **href** |
| 19 | **lang**  
Base language of an element’s attributes and text |
| 20 | **maxlength**  
Maximum number of characters for text fields |
| 21 | **readonly**  
Read-only state of an input field; text can be selected in a readonly field but not edited |
| 22 | **rel**  
Relationship between the current document and a link specified with the **href** attribute |
| 23 | **rev**  
Reverse link from the anchor specified with **href** to the current document. The value of the attribute is a space-separated list of link types |
<table>
<thead>
<tr>
<th>24</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of visible rows in a text area. <strong>h:dataTable</strong> has a <strong>rows</strong> attribute, but it’s not an HTML pass-through attribute</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>25</th>
<th>shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape of a region. Valid values: <strong>default</strong>, <strong>rect</strong>, <strong>circle</strong>, <strong>poly</strong>. (default signifies the entire region)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>26</th>
<th>style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inline style information</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>27</th>
<th>tabindex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical value specifying a tab index</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>28</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name of a frame in which a document is opened</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>29</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>30</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of a link; for example, <strong>stylesheet</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width of an element</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32</th>
<th>onblur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element loses focus</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>33</th>
<th>onchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element’s value changes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Event</td>
</tr>
<tr>
<td>---</td>
<td>-----------------</td>
</tr>
<tr>
<td>34</td>
<td>onclick</td>
</tr>
<tr>
<td>35</td>
<td>ondblclick</td>
</tr>
<tr>
<td>36</td>
<td>onfocus</td>
</tr>
<tr>
<td>37</td>
<td>onkeydown</td>
</tr>
<tr>
<td>38</td>
<td>onkeypress</td>
</tr>
<tr>
<td>39</td>
<td>onkeyup</td>
</tr>
<tr>
<td>40</td>
<td>onmousedown</td>
</tr>
<tr>
<td>41</td>
<td>onmousemove</td>
</tr>
<tr>
<td>42</td>
<td>onmouseout</td>
</tr>
<tr>
<td>43</td>
<td>onmouseover</td>
</tr>
<tr>
<td>44</td>
<td>onmouseup</td>
</tr>
<tr>
<td>45</td>
<td>onreset</td>
</tr>
</tbody>
</table>
Text is selected in an input field

Example Application
Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>result.xhtml</code> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>UserData.java</code> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

**home.xhtml**

```xml
<%xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<head>
    <title>JSF Tutorial!</title>
</head>
<h:body>
```
<h2>h::selectOneMenu example</h2>
<hr />
<h:form>
<h3>Combo Box</h3>
<h:selectOneMenu value="#{userData.data}"
    <f:selectItem itemValue="1" itemLabel="Item 1" />
    <f:selectItem itemValue="2" itemLabel="Item 2" />
    <f:selectItem itemValue="3" itemLabel="Item 3" />
    <f:selectItem itemValue="4" itemLabel="Item 4" />
    <f:selectItem itemValue="5" itemLabel="Item 5" />
</h:selectOneMenu>
<h:commandButton value="Submit" action="result" />
</h:form>
</h:body>
</html>

result.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf.facelets">
<head>
    <title>JSF Tutorial!</title>
</head>
<h:body>
    <h2>Result</h2>
    <hr />
    #{userData.data}
</h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Select any option and press **Submit** button. We've selected item 4. You will see the selected results.
**h:outputText**

The `h:outputText` tag renders an HTML text.

**JSF Tag**

```xml
<h:outputText value="Hello World!" />
```

**Rendered Output**

```
Hello World!
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>7</td>
<td>style</td>
</tr>
<tr>
<td></td>
<td>Inline style information</td>
</tr>
</tbody>
</table>
A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title's value.

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - Create Application chapter. If everything is fine with your application, this will produce the following result.

**h:outputFormat**

The `h:outputFormat` tag renders an HTML text but can accept parameterised inputs.

**JSF Tag**

```xml
<h:outputFormat value="parameter 1 : {0}, parameter 2 : {1}" >
   <f:param value="Item 1" />
   <f:param value="Item 2" />
</h:outputFormat>
```
### Rendered Output

| parameter 1 | Item 1 | parameter 2 | Item 2 |

### Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>id</strong></td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td><strong>binding</strong></td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td><strong>rendered</strong></td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td><strong>styleClass</strong></td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td><strong>value</strong></td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td><strong>converter</strong></td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>7</td>
<td><strong>style</strong></td>
</tr>
<tr>
<td></td>
<td>Inline style information</td>
</tr>
<tr>
<td>8</td>
<td><strong>title</strong></td>
</tr>
<tr>
<td></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

*home.xhtml*

```xml
<!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">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <body>
    <h2>h:outputFormat example</h2>
    <hr />
    <h:form>
      <h3>Text</h3>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

**h:graphicImage**

The `h:graphicImage` tag renders an HTML element of the type "img".

**JSF Tag**

```html
<h:graphicImage value="http://www.tutorialspoint.com/images/jsf-mini-logo.png"/>
```

**Rendered Output**

```html
<img src="http://www.tutorialspoint.com/images/jsf-mini-logo.png"/>
```
# Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>alt</td>
</tr>
<tr>
<td></td>
<td>Alternative text for nontextual elements such as images or applets</td>
</tr>
<tr>
<td>7</td>
<td>dir</td>
</tr>
<tr>
<td></td>
<td>Direction for text. Valid values are ltr (left to right) and rtl (right to left)</td>
</tr>
<tr>
<td>8</td>
<td>lang</td>
</tr>
<tr>
<td></td>
<td>Base language of an element’s attributes and text</td>
</tr>
<tr>
<td>9</td>
<td>style</td>
</tr>
<tr>
<td></td>
<td>Inline style information</td>
</tr>
<tr>
<td>10</td>
<td>tabindex</td>
</tr>
<tr>
<td></td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>11</td>
<td>target</td>
</tr>
<tr>
<td></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 12 | **title**  
A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value |
| 13 | **usemap**  
Usemap of an element |
| 14 | **url**  
Url of the image |
| 15 | **width**  
Width of an element |
| 16 | **onblur**  
Element loses focus |
| 17 | **onchange**  
Element’s value changes |
| 18 | **onclick**  
Mouse button is clicked over the element |
| 19 | **ondblclick**  
Mouse button is double-clicked over the element |
| 20 | **onfocus**  
Element receives focus |
| 21 | **onkeydown**  
Key is pressed |
| 22 | **onkeypress**  
Key is pressed and subsequently released |
| 23 | **onkeyup**  
Key is released |
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 24 | `onmousedown`  
Mouse button is pressed over the element |
| 25 | `onmousemove`  
Mouse moves over the element |
| 26 | `onmouseout`  
Mouse leaves the element’s area |
| 27 | `onmouseover`  
Mouse moves onto an element |
| 28 | `onmouseup`  
Mouse button is released |

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
</head>
```
<h:graphicImage example/>

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

**h:outputStylesheet**

The h:outputStylesheet tag renders an HTML element of the type "link" with type "text/css". This tag is used to add external stylesheet file to JSF page.

**JSF Tag**

```
<h:outputStylesheet library="css" name="styles.css"/>
```
### Rendered Output

```html
<link type="text/css" rel="stylesheet"
href="/helloworld/javax.faces.resource/styles.css.jsf?ln=css" />
```

### Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create <code>resources</code> folder under <code>src &gt; main</code> folder.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>css</code> folder under <code>src &gt; main &gt; resources</code> folder.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>styles.css</code> file under <code>src &gt; main &gt; resources &gt; css</code> folder.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <code>styles.css</code> file as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Modify <code>pom.xml</code> as explained below.</td>
</tr>
<tr>
<td>7</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>8</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>9</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
</tbody>
</table>
Launch your web application using appropriate URL as explained below in the last step.

styles.css

```css
.message{
    color:green;
}
```

pom.xml

```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.tutorialspoint.test</groupId>
    <artifactId>helloworld</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>helloworld 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>2.1.7</version>
        </dependency>
        <dependency>
            <groupId>com.sun.faces</groupId>
            <artifactId>jsf-impl</artifactId>
            <version>2.1.7</version>
        </dependency>
    </dependencies>
</project>
```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<build>
  <finalName>helloworld</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-resources-plugin</artifactId>
      <version>2.6</version>
      <executions>
        <execution>
          <id>copy-resources</id>
          <phase>validate</phase>
          <goals>
            <goal>copy-resources</goal>
          </goals>
          <configuration>
            <outputDirectory>${basedir}/target/helloworld/resources</outputDirectory>
            <resources>
              <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
              </resource>
            </resources>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
h:outputScript

The h:outputScript tag renders an HTML element of the type "script" with type "text/javascript". This tag is used to add an external javascript file to JSF page.

**JSF Tag**

```xml
<h:outputScript library="js" name="help.js" />
```

**Rendered Output**

```html
<script type="text/javascript"
    src="/helloworld/javax.faces.resource/help.js.jsf?ln=js"></script>
```

**Example Application**

Let us create a test JSF application to test the above tag.
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create <code>resources</code> folder under <code>src &gt; main</code> folder.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>js</code> folder under <code>src &gt; main &gt; resources</code> folder.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>help.js</code> file under <code>src &gt; main &gt; resources &gt; js</code> folder.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <code>help.js</code> file as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Modify <code>pom.xml</code> as explained below.</td>
</tr>
<tr>
<td>7</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>8</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
</tbody>
</table>
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**help.js**

```javascript
function showMessage(){
    alert("Hello World!");
}
```

**pom.xml**

```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.tutorialspoint.test</groupId>
    <artifactId>helloworld</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>helloworld 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>2.1.7</version>
</dependency>

<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.1.7</version>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<build>
  <finalName>helloworld</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-resources-plugin</artifactId>
      <version>2.6</version>
      <executions>
        <execution>
          <id>copy-resources</id>
          <phase>validate</phase>
          <goals>
            <goal>copy-resources</goal>
          </goals>
          <configuration>
            <outputDirectory>${basedir}/target/helloworld/resources
<outputDirectory>
<resources>
<resource>
<directory>src/main/resources</directory>
<filtering>true</filtering>
</resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>

/home.xhtml

<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<h:head>
<title>JSF Tutorial!</title>
<h:outputScript library="js" name="help.js" />
</h:head>
<h:body>
<h:outputScript example</h2>
<h1>h:commandButton onclick="showMessage();" />
</h:form>
</h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

**h:commandButton**

The `h:commandButton` tag renders an HTML input element of the type "submit".

**JSF Tag**

```xml
<h:commandButton value="Click Me!" onclick="alert('Hello World!');" />
```

**Rendered Output**

```html
<input type="submit" name="j_idt10:j_idt13" value="Click Me!"
   onclick="alert('Hello World!');" />
```

**Tag Attributes**
<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>3</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>4</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>5</td>
<td>coords</td>
</tr>
<tr>
<td></td>
<td>Coordinates for an element whose shape is a rectangle, circle, or polygon</td>
</tr>
<tr>
<td>6</td>
<td>dir</td>
</tr>
<tr>
<td></td>
<td>Direction for text. Valid values are ltr (left to right) and rtl (right to left)</td>
</tr>
<tr>
<td>7</td>
<td>disabled</td>
</tr>
<tr>
<td></td>
<td>Disabled state of an input element or button</td>
</tr>
<tr>
<td>8</td>
<td>tabindex</td>
</tr>
<tr>
<td></td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>9</td>
<td>target</td>
</tr>
<tr>
<td></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>10</td>
<td>title</td>
</tr>
<tr>
<td></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>11</td>
<td>width</td>
</tr>
<tr>
<td></td>
<td>Event</td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td>12</td>
<td>onblur</td>
</tr>
<tr>
<td>13</td>
<td>onchange</td>
</tr>
<tr>
<td>14</td>
<td>onclick</td>
</tr>
<tr>
<td>15</td>
<td>ondblclick</td>
</tr>
<tr>
<td>16</td>
<td>onfocus</td>
</tr>
<tr>
<td>17</td>
<td>onkeydown</td>
</tr>
<tr>
<td>18</td>
<td>onkeypress</td>
</tr>
<tr>
<td>19</td>
<td>onkeyup</td>
</tr>
<tr>
<td>20</td>
<td>onmousedown</td>
</tr>
<tr>
<td>21</td>
<td>onmousemove</td>
</tr>
<tr>
<td>22</td>
<td>onmouseout</td>
</tr>
</tbody>
</table>
### Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
</tbody>
</table>
Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```xml
<!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">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <body>
    <h2>h:commandButton example</h2>
    <hr />
    <h:form>
      <h:commandButton value="Click Me!" onclick="alert('Hello World!');" />
    </h:form>
  </body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
**h:Link**

The `h:Link` tag renders an HTML "anchor" element.

**JSF Tag**

```xml
<h:link value="Page 1" outcome="page1" />
```
### Rendered Output

```html
<a href="/helloworld/page1.jsf">Page 1</a>
```

### Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component’s value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>A method binding to a method that responds to value changes</td>
</tr>
<tr>
<td>7</td>
<td>converter</td>
</tr>
<tr>
<td></td>
<td>Converter class name</td>
</tr>
<tr>
<td>8</td>
<td>validator</td>
</tr>
<tr>
<td></td>
<td>Class name of a validator that’s created and attached to a component</td>
</tr>
<tr>
<td>9</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>A boolean; if true, requires a value to be entered in the associated field</td>
</tr>
<tr>
<td>10</td>
<td>accesskey</td>
</tr>
<tr>
<td></td>
<td>A key, typically combined with a system-defined metakey, that gives focus to an element</td>
</tr>
<tr>
<td>11</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td>Comma-separated list of content types for a form</td>
</tr>
<tr>
<td></td>
<td>Tag</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>12</td>
<td>accept-chars</td>
</tr>
<tr>
<td>13</td>
<td>alt</td>
</tr>
<tr>
<td>14</td>
<td>border</td>
</tr>
<tr>
<td>15</td>
<td>charset</td>
</tr>
<tr>
<td>16</td>
<td>coords</td>
</tr>
<tr>
<td>17</td>
<td>dir</td>
</tr>
<tr>
<td>18</td>
<td>hreflang</td>
</tr>
<tr>
<td>19</td>
<td>lang</td>
</tr>
<tr>
<td>20</td>
<td>maxlength</td>
</tr>
<tr>
<td>21</td>
<td>readonly</td>
</tr>
<tr>
<td>22</td>
<td>rel</td>
</tr>
<tr>
<td>23</td>
<td>rev</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>Size of an input field</td>
</tr>
<tr>
<td><strong>style</strong></td>
<td>Inline style information</td>
</tr>
<tr>
<td><strong>tabindex</strong></td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td><strong>target</strong></td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td><strong>title</strong></td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Type of a link; for example, <strong>stylesheet</strong></td>
</tr>
<tr>
<td><strong>width</strong></td>
<td>Width of an element</td>
</tr>
<tr>
<td><strong>onblur</strong></td>
<td>Element loses focus</td>
</tr>
<tr>
<td><strong>onchange</strong></td>
<td>Element’s value changes</td>
</tr>
<tr>
<td><strong>onclick</strong></td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td><strong>ondblclick</strong></td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td><strong>onfocus</strong></td>
<td>Element receives focus</td>
</tr>
<tr>
<td></td>
<td>Event</td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
</tr>
<tr>
<td>37</td>
<td><code>onkeydown</code></td>
</tr>
<tr>
<td>38</td>
<td><code>onkeypress</code></td>
</tr>
<tr>
<td>39</td>
<td><code>onkeyup</code></td>
</tr>
<tr>
<td>39</td>
<td><code>onmousedown</code></td>
</tr>
<tr>
<td>40</td>
<td><code>onmousemove</code></td>
</tr>
<tr>
<td>41</td>
<td><code>onmouseout</code></td>
</tr>
<tr>
<td>42</td>
<td><code>onmouseover</code></td>
</tr>
<tr>
<td>43</td>
<td><code>onmouseup</code></td>
</tr>
<tr>
<td>44</td>
<td><code>onreset</code></td>
</tr>
<tr>
<td>45</td>
<td><code>onselect</code></td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <body>
    <h2>h:Link example</h2>
    <hr />
    <h:form>
      <h:link value="Page 1" outcome="page1" />
    </h:form>
  </body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in *JSF - First Application* chapter. If everything is fine with your application, this will produce the following result.
**h:commandLink**

The `h:commandLink` tag renders an HTML "anchor" element.

**JSF Tag**

```xml
<h:commandLink value="Page 1" action="page1"/>
```

**Rendered Output**

```html
<a href="#" onclick="mojarra.jsfcljs(document.getElementById('j_idt13'),
{"j_idt13:j_idt14':"j_idt13:j_idt14"},'');return false">Page 1</a>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>id</strong> Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td><strong>binding</strong> Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td><strong>rendered</strong> A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td><strong>styleClass</strong> Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 5 | **value**  
A component’s value, typically a value binding |   |
| 6 | **valueChangeListener**  
A method binding to a method that responds to value changes |   |
| 7 | **converter**  
Converter class name |   |
| 8 | **validator**  
Class name of a validator that’s created and attached to a component |   |
| 9 | **required**  
A boolean; if true, requires a value to be entered in the associated field |   |
| 10 | **Accesskey**  
A key, typically combined with a system-defined metakey, that gives focus to an element |   |
| 11 | **accept**  
Comma-separated list of content types for a form |   |
| 12 | **accept-charset**  
Comma- or space-separated list of character encodings for a form. The **accept-charset** attribute is specified with the JSF HTML attribute named **acceptcharset** |   |
| 13 | **Alt**  
Alternative text for nontextual elements such as images or applets |   |
| 14 | **border**  
Pixel value for an element’s border width |   |
| 15 | **charset**  
Character encoding for a linked resource |   |
| 16 | **coords**  
Coordinates for an element whose shape is a rectangle, circle, or polygon |
| 17 | **dir**  
Direction for text. Valid values are ltr (left to right) and rtl (right to left) |
| 18 | **hreflang**  
Base language of a resource specified with the href attribute; hreflang may only be used with href |
| 19 | **lang**  
Base language of an element’s attributes and text |
| 20 | **maxlength**  
Maximum number of characters for text fields |
| 21 | **readonly**  
Read-only state of an input field; text can be selected in a readonly field but not edited |
| 22 | **rel**  
Relationship between the current document and a link specified with the href attribute |
| 23 | **rev**  
Reverse link from the anchor specified with href to the current document. The value of the attribute is a space-separated list of link types |
| 24 | **size**  
Size of an input field |
| 25 | **style**  
Inline style information |
<table>
<thead>
<tr>
<th></th>
<th>attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>tabindex</td>
<td>Numerical value specifying a tab index</td>
</tr>
<tr>
<td>27</td>
<td>target</td>
<td>The name of a frame in which a document is opened</td>
</tr>
<tr>
<td>28</td>
<td>title</td>
<td>A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value</td>
</tr>
<tr>
<td>29</td>
<td>type</td>
<td>Type of a link; for example, stylesheet</td>
</tr>
<tr>
<td>30</td>
<td>width</td>
<td>Width of an element</td>
</tr>
<tr>
<td>31</td>
<td>onblur</td>
<td>Element loses focus</td>
</tr>
<tr>
<td>32</td>
<td>onchange</td>
<td>Element’s value changes</td>
</tr>
<tr>
<td>33</td>
<td>onclick</td>
<td>Mouse button is clicked over the element</td>
</tr>
<tr>
<td>34</td>
<td>ondblclick</td>
<td>Mouse button is double-clicked over the element</td>
</tr>
<tr>
<td>35</td>
<td>onfocus</td>
<td>Element receives focus</td>
</tr>
<tr>
<td></td>
<td>Event</td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>36</td>
<td>onkeydown</td>
<td>Key is pressed</td>
</tr>
<tr>
<td>37</td>
<td>onkeypress</td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td>38</td>
<td>onkeyup</td>
<td>Key is released</td>
</tr>
<tr>
<td>39</td>
<td>onmousedown</td>
<td>Mouse button is pressed over the element</td>
</tr>
<tr>
<td>40</td>
<td>onmousemove</td>
<td>Mouse moves over the element</td>
</tr>
<tr>
<td>41</td>
<td>onmouseout</td>
<td>Mouse leaves the element’s area</td>
</tr>
<tr>
<td>42</td>
<td>onmouseover</td>
<td>Mouse moves onto an element</td>
</tr>
<tr>
<td>43</td>
<td>onmouseup</td>
<td>Mouse button is released</td>
</tr>
<tr>
<td>44</td>
<td>onreset</td>
<td>Form is reset</td>
</tr>
<tr>
<td>45</td>
<td>onselect</td>
<td>Text is selected in an input field</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>hello world</em> under a package <code>com.tutorialspoint.test</code> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>`h:commandLink` example</h2>
    <hr />
    <h:form>
        <h:commandLink value="Page 1" action="page1" />
    </h:form>
</body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

**h:outputLink**

The `h:outputLink` tag renders an HTML "anchor" element.

**JSF Tag**

```xml
<h:outputLink value="page1.jsf">Page 1</h:outputLink>
```

**Rendered Output**

```html
<a href="page1.jsf">Page 1</a>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td>Reference to the component that can be used in a backing bean</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>3 \textit{rendered} \newline \quad A boolean; false suppresses rendering</td>
<td></td>
</tr>
<tr>
<td>4 \textit{styleClass} \newline \quad Cascading stylesheet (CSS) class name</td>
<td></td>
</tr>
<tr>
<td>5 \textit{value} \newline \quad A component’s value, typically a value binding</td>
<td></td>
</tr>
<tr>
<td>6 \textit{valueChangeListener} \newline \quad A method binding to a method that responds to value changes</td>
<td></td>
</tr>
<tr>
<td>7 \textit{converter} \newline \quad Converter class name</td>
<td></td>
</tr>
<tr>
<td>8 \textit{validator} \newline \quad Class name of a validator that’s created and attached to a component</td>
<td></td>
</tr>
<tr>
<td>9 \textit{required} \newline \quad A boolean; if true, requires a value to be entered in the associated field</td>
<td></td>
</tr>
<tr>
<td>10 \textit{accesskey} \newline \quad A key, typically combined with a system-defined metakey, that gives focus to an element</td>
<td></td>
</tr>
<tr>
<td>11 \textit{accept} \newline \quad Comma-separated list of content types for a form</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Attribute</td>
</tr>
<tr>
<td>---</td>
<td>-----------</td>
</tr>
<tr>
<td>12</td>
<td>accept-charset</td>
</tr>
<tr>
<td>13</td>
<td>alt</td>
</tr>
<tr>
<td>14</td>
<td>border</td>
</tr>
<tr>
<td>15</td>
<td>charset</td>
</tr>
<tr>
<td>16</td>
<td>coords</td>
</tr>
<tr>
<td>17</td>
<td>dir</td>
</tr>
<tr>
<td>18</td>
<td>hreflang</td>
</tr>
<tr>
<td>19</td>
<td>lang</td>
</tr>
<tr>
<td>20</td>
<td>maxlength</td>
</tr>
<tr>
<td>21</td>
<td>readonly</td>
</tr>
<tr>
<td>Index</td>
<td>Attribute</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>22</td>
<td>rel</td>
</tr>
<tr>
<td>23</td>
<td>rev</td>
</tr>
<tr>
<td>24</td>
<td>size</td>
</tr>
<tr>
<td>25</td>
<td>style</td>
</tr>
<tr>
<td>26</td>
<td>tabindex</td>
</tr>
<tr>
<td>27</td>
<td>target</td>
</tr>
<tr>
<td>28</td>
<td>title</td>
</tr>
<tr>
<td>29</td>
<td>type</td>
</tr>
<tr>
<td>30</td>
<td>width</td>
</tr>
<tr>
<td>31</td>
<td>onblur</td>
</tr>
<tr>
<td></td>
<td>Event Type</td>
</tr>
<tr>
<td>---</td>
<td>------------</td>
</tr>
<tr>
<td>32</td>
<td>onchange</td>
</tr>
<tr>
<td>33</td>
<td>onclick</td>
</tr>
<tr>
<td>34</td>
<td>ondblclick</td>
</tr>
<tr>
<td>35</td>
<td>onfocus</td>
</tr>
<tr>
<td>36</td>
<td>onkeydown</td>
</tr>
<tr>
<td>37</td>
<td>onkeypress</td>
</tr>
<tr>
<td>38</td>
<td>onkeyup</td>
</tr>
<tr>
<td>39</td>
<td>onmousedown</td>
</tr>
<tr>
<td>40</td>
<td>onmousemove</td>
</tr>
<tr>
<td>41</td>
<td>onmouseout</td>
</tr>
<tr>
<td>42</td>
<td>onmouseover</td>
</tr>
<tr>
<td>43</td>
<td>onmouseup</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify home.xhtml as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

home.xhtml

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
</head>
<body>
  <h2><h:outputLink example></h2>
  <hr />
  <h:form>
    <h:outputLink value="page1.jsf" >Page 1</h:outputLink>
  </h:form>
</body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![](image-url)

**h:panelGrid**

The h:panel tag renders an HTML "table" element.

**JSF Tag**

```xml
<h:panelGrid id="panel" columns="2" border="1"
    cellpadding="10" cellspacing="1">
    <f:facet name="header">
        <h:outputText value="Login"/>
    </f:facet>
    <h:outputLabel value="Username"/>
    <h:inputText />
    <h:outputLabel value="Password"/>
    <h:inputSecret />
    <f:facet name="footer">
        <h:panelGroup style="display:block; text-align:center">
            <h:commandButton id="submit" value="Submit" />
        </h:panelGroup>
    </f:facet>
</h:panelGrid>
```

**Rendered Output**

```xml
<table id="j_idt10:panel" border="1" cellpadding="10" cellspacing="1">
<thead>
    <tr><th colspan="2" scope="colgroup">Login</th></tr>
</thead>
</table>
```
<table>
  <tfoot>
    <tr>
      <td colspan="2">
        <span style="display:block; text-align:center">
          <input id="j_idt10:submit" type="submit"
            name="j_idt10:submit" value="Submit" />
        </span>
      </td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td><label>Username</label></td>
      <td><input type="text" name="j_idt10:j_idt17" /></td>
    </tr>
    <tr>
      <td><label>Password</label></td>
      <td><input type="password" name="j_idt10:j_idt21" value="" /></td>
    </tr>
  </tbody>
</table>

Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component's value, typically a value binding</td>
</tr>
<tr>
<td>6</td>
<td>bgcolor</td>
</tr>
<tr>
<td></td>
<td>Background color for the table</td>
</tr>
<tr>
<td></td>
<td><strong>Property</strong></td>
</tr>
<tr>
<td>---</td>
<td>--------------</td>
</tr>
<tr>
<td>7</td>
<td><code>border</code></td>
</tr>
<tr>
<td>8</td>
<td><code>cellpadding</code></td>
</tr>
<tr>
<td>9</td>
<td><code>cellspacing</code></td>
</tr>
<tr>
<td>10</td>
<td><code>columnClasses</code></td>
</tr>
<tr>
<td>11</td>
<td><code>columns</code></td>
</tr>
<tr>
<td>12</td>
<td><code>footerClass</code></td>
</tr>
<tr>
<td>13</td>
<td><code>frame</code></td>
</tr>
<tr>
<td>14</td>
<td><code>headerClass</code></td>
</tr>
<tr>
<td>15</td>
<td><code>rowClasses</code></td>
</tr>
<tr>
<td>16</td>
<td><code>rules</code></td>
</tr>
<tr>
<td>17</td>
<td><code>summary</code></td>
</tr>
<tr>
<td></td>
<td><strong>Attribute</strong></td>
</tr>
<tr>
<td>---</td>
<td>---------------</td>
</tr>
<tr>
<td>18</td>
<td><strong>dir</strong></td>
</tr>
<tr>
<td>19</td>
<td><strong>lang</strong></td>
</tr>
<tr>
<td>20</td>
<td><strong>border</strong></td>
</tr>
<tr>
<td>21</td>
<td><strong>title</strong></td>
</tr>
<tr>
<td>22</td>
<td><strong>width</strong></td>
</tr>
<tr>
<td>23</td>
<td><strong>onblur</strong></td>
</tr>
<tr>
<td>24</td>
<td><strong>onchange</strong></td>
</tr>
<tr>
<td>25</td>
<td><strong>onclick</strong></td>
</tr>
<tr>
<td>26</td>
<td><strong>ondblclick</strong></td>
</tr>
<tr>
<td>27</td>
<td><strong>onfocus</strong></td>
</tr>
<tr>
<td></td>
<td>Event Name</td>
</tr>
<tr>
<td>---</td>
<td>-----------------</td>
</tr>
<tr>
<td>28</td>
<td>onkeydown</td>
</tr>
<tr>
<td>29</td>
<td>onkeypress</td>
</tr>
<tr>
<td>30</td>
<td>onkeyup</td>
</tr>
<tr>
<td>31</td>
<td>onmousedown</td>
</tr>
<tr>
<td>32</td>
<td>onmousemove</td>
</tr>
<tr>
<td>33</td>
<td>onmouseout</td>
</tr>
<tr>
<td>34</td>
<td>onmouseover</td>
</tr>
<tr>
<td>35</td>
<td>onmouseup</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
</head>
<body>
  <h2>h:panelGrid example</h2>
  <hr />
  <h:form>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
**h:message**

The h:message tag displays message corresponding to UI element.

**JSF Tag**

```
<h:inputText id="username" size="20" label="UserName" required="true">
    <f:validateLength for="username" minimum="5" maximum="20" />
</h:inputText>
<h:message for="username" style="color:red" />
```

**Rendered Output**

In case the username entered is more than 20 characters.

```
<span style="color:red">UserName: Validation Error:
Length is greater than allowable maximum of '20'</span>
```

In case the username entered is less than 5 characters.

```
<span style="color:red">UserName: Validation Error:
Length is less than allowable minimum of '5'</span>
```

In case the username is not entered.

```
<span style="color:red">UserName: Validation Error:
Value is required</span>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td>5</td>
<td>for</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>6</td>
<td>errorClass</td>
</tr>
<tr>
<td>7</td>
<td>errorStyle</td>
</tr>
<tr>
<td>8</td>
<td>fatalClass</td>
</tr>
<tr>
<td>9</td>
<td>fatalStyle</td>
</tr>
<tr>
<td>10</td>
<td>globalOnly</td>
</tr>
<tr>
<td>11</td>
<td>infoClass</td>
</tr>
<tr>
<td>12</td>
<td>infoStyle</td>
</tr>
<tr>
<td>13</td>
<td>layout</td>
</tr>
<tr>
<td>14</td>
<td>showDetail</td>
</tr>
<tr>
<td>15</td>
<td>showSummary</td>
</tr>
</tbody>
</table>
### Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <code>com.tutorialspoint.test</code> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
</tbody>
</table>
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

home.xhtml

```xml
<!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">
  <head>
    <title>JSF Tutorial!</title>
  </head>
  <body>
    <h2>h:messages example</h2>
    <hr />
    <h:form>
      <h:panelGrid id="panel" columns="3" border="0" cellspacing="10" cellpadding="10">
        <h:outputLabel value="Enter Username" />
        <h:inputText id="username" size="20" label="UserName"
          required="true">
          <f:validateLength for="username" minimum="5" maximum="20" />
        </h:inputText>
        <h:message for="username" style="color:red" />
        <h:outputLabel value="Enter Password" />
        <h:inputSecret id="password" size="20" label="Password"
          required="true" redisplay="true" >
          <f:validateLength for="password" minimum="5" maximum="10" />
        </h:inputSecret>
        <h:message for="password" style="color:red" />
        <h:commandButton id="submit" value="Submit" action="result" />
      </h:panelGrid>
    </h:form>
  </body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![h:messages example](image)

**h:messages**
The `h:messages` tag shows all the messages at one place corresponding to UI elements.

**JSF Tag**

```xml
<h:messages style="color:red;margin:8px;" />
```

**Rendered Output**

Case: Username entered is more than 20 characters and password entered is less than 5 characters.

```html
<ul style="color:red;margin:8px;">
    <li>User Name: Validation Error: Length is greater than allowable maximum of '20'</li>
    <li>Password: Validation Error: Length is less than allowable minimum of '5'</li>
</ul>
```
## Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>for</td>
</tr>
<tr>
<td></td>
<td>The ID of the component whose message is displayed, applicable only to h:message</td>
</tr>
<tr>
<td>6</td>
<td>errorClass</td>
</tr>
<tr>
<td></td>
<td>CSS class applied to error messages</td>
</tr>
<tr>
<td>7</td>
<td>errorStyle</td>
</tr>
<tr>
<td></td>
<td>CSS style applied to error messages</td>
</tr>
<tr>
<td>8</td>
<td>fatalClass</td>
</tr>
<tr>
<td></td>
<td>CSS class applied to fatal messages</td>
</tr>
<tr>
<td>9</td>
<td>fatalStyle</td>
</tr>
<tr>
<td></td>
<td>CSS style applied to fatal messages</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>globalOnly</strong></td>
<td>Instruction to display only global messages, applicable only to h:messages. Default: false</td>
</tr>
<tr>
<td><strong>infoClass</strong></td>
<td>CSS class applied to information messages</td>
</tr>
<tr>
<td><strong>infoStyle</strong></td>
<td>CSS style applied to information messages</td>
</tr>
<tr>
<td><strong>layout</strong></td>
<td>Specification for message layout: table or list, applicable only to h:messages</td>
</tr>
<tr>
<td><strong>showDetail</strong></td>
<td>A boolean that determines whether message details are shown. Defaults are false for h:messages, true for h:message</td>
</tr>
<tr>
<td><strong>showSummary</strong></td>
<td>A boolean that determines whether message summaries are shown. Defaults are true for h:messages, false for h:message</td>
</tr>
<tr>
<td><strong>tooltip</strong></td>
<td>A boolean that determines whether message details are rendered in a tooltip; the tooltip is only rendered if showDetail and showSummary are true</td>
</tr>
<tr>
<td><strong>warnClass</strong></td>
<td>CSS class for warning messages</td>
</tr>
<tr>
<td><strong>warnStyle</strong></td>
<td>CSS style for warning messages</td>
</tr>
<tr>
<td><strong>style</strong></td>
<td>Inline style information</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<!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">
  <head>
    <title>JSF Tutorial!</title>
  </head>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
**f:param**

`f:param` tag provides the options to pass parameters to a component or pass request parameters.

**JSF Tag**

Pass parameter to a UI component

```xml
<h:outputFormat value="Hello {0}!.">
   <f:param value="World" />
</h:outputFormat>
```

Pass request parameter

```xml
<h:commandButton id="submit"
   value="Show Message" action="#{userData.showResult}"
   >
   <f:param name="username" value="JSF 2.0 User" />
</h:commandButton>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>binding</td>
</tr>
<tr>
<td></td>
<td>Reference to the component that can be used in a backing bean</td>
</tr>
<tr>
<td>3</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>An optional name for this parameter component</td>
</tr>
<tr>
<td>4</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>The value stored in this component</td>
</tr>
</tbody>
</table>
**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <strong>helloworld</strong> under a package <code>com.tutorialspoint.test</code> as explained in the <strong>JSF - First Application</strong> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <strong>home.xhtml</strong> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <strong>result.xhtml</strong> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <strong>UserData.java</strong> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public String showResult(){
        FacesContext fc = FacesContext.getCurrentInstance();
        Map<String,String> params =
        fc.getExternalContext().getRequestParameterMap();
        data = params.get("username");
        return "result";
    }
}

home.xhtml

<!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">
<head>
    <title>JSF Tutorial!</title>
</head>
<body>
    <h2>f:param example</h2>
    <hr />

    tutorialspoint
    SIMPLY EASY LEARNING
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Press **Show Message** button and you'll see the following result.

---

**f:attribute**

The `h:attribute` tag provides option to pass a attribute value to a component, or a parameter to a component via action listener.

**JSF Tag**

```xml
<h:commandButton id="submit"
    actionListener="#{userData.attributeListener}" action="result">
    <f:attribute name="value" value="Show Message" />
    <f:attribute name="username" value="JSF 2.0 User" />
</h:commandButton>
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>The name of the attribute to set</td>
</tr>
<tr>
<td>2</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>The value of the attribute</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <em>UserData.java</em> as a managed bean under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;
```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {
    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

    public void attributeListener(ActionEvent event) {
        data = (String) event.getComponent().getAttributes().get("username");
    }
}

home.xhtml

<!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">
    <head>
        <title>JSF Tutorial!</title>
    </head>
    <body>
        <h2>f:attribute example</h2>
        <hr />
        <h:commandButton id="submit"
            actionListener="#{userData.attributeListener}" action="result">
            <f:attribute name="value" value="Show Message" />
        </h:commandButton>
    </body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Press **Show Message** button and you'll see the following result.

![Image of a browser window showing the result](image)

**h:setPropertyActionListener**

The `h:setPropertyActionListener` tag adds an action listener to a component that sets a bean property to a given value.

**JSF Tag**

```xml
<h:commandButton id="submit" action="result" value="Show Message">
    <f:setPropertyActionListener target="#{userData.data}" value="JSF 2.0 User" />
</h:commandButton>
```

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>result.xhtml</code> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>UserData.java</code> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public String data = "1";

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```
home.xhtml

```xml
<!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">
<head>
  <title>JSF Tutorial!</title>
</head>
<body>
  <h2>f:attribute example</h2>
  <hr />
  <h:form>
    <h:commandButton id="submit" action="result" value="Show Message">
      <f:setPropertyActionListener
        target="#{userData.data}" value="JSF 2.0 User" />
    </h:commandButton>
  </h:form>
</body>
</html>
```

result.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title>JSF Tutorial!</title>
</head>
<body>
  <h2>Result</h2>
  <hr />
  #{userData.data}
</body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Press **Show Message** button and you'll see the following result.
JSF provides special tags to create common layout for a web application called facelets tags. These tags provide flexibility to manage common parts of multiple pages at one place.

For these tags, you need to use the following namespaces of URI in html node.

```html
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:ui="http://java.sun.com/jsf/facelets"
>
```

Following are important *Facelets Tags* in JSF 2.0.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong></td>
<td><strong>Templates</strong></td>
</tr>
<tr>
<td></td>
<td>We’ll demonstrate how to use templates using the following tags</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;ui:insert&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;ui:define&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;ui:include&gt;</code></td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;ui:composition&gt;</code></td>
</tr>
<tr>
<td><strong>2</strong></td>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td></td>
<td>We’ll demonstrate how to pass parameters to a template file using the following tag</td>
</tr>
<tr>
<td></td>
<td>• <code>&lt;ui:param&gt;</code></td>
</tr>
</tbody>
</table>
Custom
We’ll demonstrate how to create custom tags

Remove
We’ll demonstrate capability to remove JSF code from generated HTML page

Template Tags
Templates in a web application defines a common interface layout and style. For example, a same banner, logo in common header and copyright information in footer. JSF provides following facelet tags to provide a standard web interface layout.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>ui:insert</strong></td>
</tr>
<tr>
<td></td>
<td>Used in template file. It defines contents to be placed in a template. ui:define tag can replaced its contents.</td>
</tr>
<tr>
<td>2</td>
<td><strong>ui:define</strong></td>
</tr>
<tr>
<td></td>
<td>Defines the contents to be inserted in a template.</td>
</tr>
<tr>
<td>3</td>
<td><strong>ui:include</strong></td>
</tr>
<tr>
<td></td>
<td>Includes contents of one xhtml page into another xhtml page.</td>
</tr>
<tr>
<td>4</td>
<td><strong>ui:composition</strong></td>
</tr>
<tr>
<td></td>
<td>Loads a template using <strong>template</strong> attribute. It can also define a group of components to be inserted in xhtml page.</td>
</tr>
</tbody>
</table>
Creating Template

Creating template for a web application is a step-by-step procedure. Following are the steps to create a sample template.

Step 1: Create Header file: header.xhtml
Use `<ui:composition>` tag to define a default content of Header section.

```xml
<ui:composition>
  <h1>Default Header</h1>
</ui:composition>
```

Step 2: Create Footer file: footer.xhtml
Use `<ui:composition>` tag to define a default content of Footer section.

```xml
<ui:composition>
  <h1>Default Footer</h1>
</ui:composition>
```

Step 3: Create Content file: contents.xhtml
Use `<ui:composition>` tag to define a default content of Content section.

```xml
<ui:composition>
  <h1>Default Contents</h1>
</ui:composition>
```

Step 4: Create a Template: common.xhtml
Use `<ui:insert>` and `<ui:include>` tag to include header/footer and content file in template file. Name each section in `<ui:insert>` tag. **name** attribute of `<ui:insert>` tag will be used to replace the contents of the corresponding section.

```xml
<h:body>
  <ui:insert name="header">
    <ui:include src="header.xhtml"/>
  </ui:insert>
  <ui:insert name="content">
    <ui:include src="contents.xhtml"/>
  </ui:insert>
  <ui:insert name="footer">
    <ui:include src="footer.xhtml"/>
  </ui:insert>
</h:body>
```
Step 5a: Use Template with default contents: home.xhtml
Load common.xhtml, a template using `ui:composition` tag in any xhtml page.

```xml
<h:body>
    <ui:composition template="common.xhtml"/>
</h:body>
```

Step 5b: Use Template and set own contents: home.xhtml
Load common.xhtml, a template using `ui:composition` tag in any xhtml page. Use `ui:define` tag to override default values.

```xml
<h:body>
    <ui:composition template="templates/common.xhtml">
        <ui:define name="content">
            <h:link value="Page 1" outcome="page1"/>
            &nbsp;
            <h:link value="Page 2" outcome="page2"/>
        </ui:define>
    </ui:composition>
</h:body>
```

Example Application
Let us create a test JSF application to test the template tags in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>Create <em>templates</em> folder under <em>src &gt; main &gt; webapp</em> folder.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>header.xhtml</em>, <em>footer.xhtml</em>, <em>contents.xhtml</em> and <em>common.xhtml</em> files under <em>src &gt; main &gt; webapp &gt; templates</em> folder. Modify them as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <em>page1.xhtml</em> and <em>page2.xhtml</em> files under <em>src &gt; main &gt; webapp</em> folder. Modify them as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

### header.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <body>
        <ui:composition>
            <h1>Default Header</h1>
        </ui:composition>
    </body>
</html>
```
footer.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <body>
        <ui:composition>
            <h1>Default Footer</h1>
        </ui:composition>
    </body>
</html>
```

contents.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <body>
        <ui:composition>
            <h1>Default Content</h1>
        </ui:composition>
    </body>
</html>
```

common.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <body>
        <ui:composition>
            
        </ui:composition>
    </body>
</html>
```
```xml
<ui:define name="header">  
  <h2>Page1 header</h2>  
</ui:define>  

<ui:define name="content">  
  <ui:include src="/template/contents.xhtml" />  
</ui:define>  

<ui:define name="footer">  
  <ui:include src="/templates/footer.xhtml" />  
</ui:define>
```

**page1.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>  
<!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:ui="http://java.sun.com/jsf/facelets">  
  <h:body>  
    <ui:composition template="/templates/common.xhtml">  
      <ui:define name="header">  
        <h2>Page1 header</h2>  
      </ui:define>  
      <ui:define name="content">  
      </ui:define>  
    </ui:composition>
  </h:body>
</html>
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets">
    <ui:composition template="templates/common.xhtml">
        <ui:define name="content">
            <br/><br/>
            <h:link value="Page 1" outcome="page1" />
            <h:link value="Page 2" outcome="page2" />
            <br/><br/>
        </ui:define>
    </ui:composition>
</h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Click **Page1** link and you'll see the following result.
Or Click Page2 link and you'll see the following result.

**ui:param Tag**

Using ui:param tag, we can pass parameters to template file or an included file.

In *JSF - template tags* chapter, we've learned how to create and use template tags. We defined various section such as header, footer, content, and a template combining all the sections.

Now we'll learn -

- How to pass parameter(s) to various section of a template
- How to pass parameter(s) to a template
Parameter to Section of a Template

Create parameter : common.xhtml
Add parameter to ui:include tag. Use ui:param tag to define a parameter containing a value to be passed to Header section.

```xml
<ui:insert name="header">
  <ui:include src="/templates/header.xhtml">
    <ui:param name="defaultHeader" value="Default Header" />
  </ui:include>
</ui:insert>
```

Use parameter : header.xhtml

```xml
<ui:composition>
  <h1>${defaultHeader}</h1>
</ui:composition>
```

Parameter to Template

Create parameter : home.xhtml
Add parameter to ui:composition tag. Use ui:param tag to define a parameter containing a value to be passed to template.

```xml
<ui:composition template="templates/common.xhtml">
  <ui:param name="title" value="Home" />
</ui:composition>
```

Use parameter : common.xhtml

```xml
<h:body>
  <h2>${title}</h2>
</h:body>
```
## Example Application

Let us create a test JSF application to test the template tags in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <code>com.tutorialspoint.test</code> as explained in the <em>JSF - Templates Tag</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>header.xhtml</code>, and <code>common.xhtml</code> files under <code>src &gt; main &gt; webapp &gt; templates</code> folder. Modify them as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>4</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>5</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>6</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

### header.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  <body>
    <ui:composition>
      <h1>#{defaultHeader}</h1>
    </ui:composition>
  </body>
</html>
```
common.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets">
<h:head>
  <h2>${title}</h2>
  <div style="border-width:2px; border-color:green; border-style:solid;">
    <ui:include name="header" >
      <ui:include src="/templates/header.xhtml" >
        <ui:param name="defaultHeader" value="Default Header" />
      </ui:include>
    </ui:include>
  </div>
  <br/>
  <ui:insert name="content" >
    <ui:include src="/templates/contents.xhtml" />
  </ui:insert>
  <br/>
  <ui:insert name="footer" >
    <ui:include src="/templates/footer.xhtml" />
  </ui:insert>
</div>
</h:body>
</html>
```

home.xhtml
<?xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition template="templates/common.xhtml">
      <ui:param name="title" value="Home" />
      <ui:define name="content">
        <br/><br/>
        <h:link value="Page 1" outcome="page1" />
        <h:link value="Page 2" outcome="page2" />
        <br/><br/>
      </ui:define>
    </ui:composition>
  </h:body>
</html>

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
**Custom Tag**

JSF provides the developer with a powerful capability to define own custom tags, which can be used to render custom contents.

Defining a custom tag in JSF is a three-step process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Create a xhtml file and define contents in it using <strong>ui:composition</strong> tag</td>
</tr>
<tr>
<td>1b</td>
<td>Create a tag library descriptor (.taglib.xml file) and declares the above custom tag in it.</td>
</tr>
<tr>
<td>1c</td>
<td>Register the tag library descriptor in web.xml</td>
</tr>
</tbody>
</table>

**Step 1a: Define custom tag contents : buttonPanel.xhtml**

```xml
<h:body>
    <ui:composition>
        <h:commandButton type="submit" value="#{okLabel}" />
        <h:commandButton type="reset" value="#{cancelLabel}" />
    </ui:composition>
</h:body>
```

**Step 1b: Define a tag library : tutorialspoint.taglib.xml**

As the name mentions a Tag library is a library of tags. Following table describes important attributes of a tag library.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Node &amp; Description</th>
</tr>
</thead>
</table>
| 1       | **facelet-taglib**  
Contains all the tags. |
| 2       | **namespace**  
Namespace of the tag library and should be unique. |
Step 1c: Register the tag library :web.xml

Using a custom tag in JSF is a two-step process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2a</td>
<td>Create a xhtml file and use custom tag library’s namespace</td>
</tr>
<tr>
<td>2b</td>
<td>Use the custom tag as normal JSF tags</td>
</tr>
</tbody>
</table>
Step 2a: Use Custom Namespace: home.xhtml

```xml
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:ui="http://java.sun.com/jsf/facelets">
    xmlns:tp="http://tutorialspoint.com/facelets">
```

Step 2b: Use Custom Tag: home.xhtml

```xml
<h:body>
   <tp:buttonPanel okLabel="Ok" cancelLabel="Cancel"/>
</h:body>
```

Example Application

Let us create a test JSF application to test the template tags in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create com folder under WEB-INF directory.</td>
</tr>
<tr>
<td>3</td>
<td>Create tutorialspoint folder under WEB-INF &gt; com directory.</td>
</tr>
<tr>
<td>4</td>
<td>Create buttonPanel.xhtml file under WEB-INF &gt; com &gt; tutorialspoint folder. Modify it as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Create tutorialspoint.taglib.xml file under WEB-INF folder. Modify it as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Modify web.xml file under WEB-INF folder as explained below.</td>
</tr>
<tr>
<td>No.</td>
<td>Step</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>7</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>8</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>9</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>10</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**buttonPanel.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets">
  <h:body>
    <ui:composition>
      <h:commandButton type="submit" value="#{okLabel}" />
      <h:commandButton type="reset" value="#{cancelLabel}" />
    </ui:composition>
  </h:body>
</html>
```

**tutorialspoint.taglib.xml**
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
"http://java.sun.com/dtd/facelet-taglib_1_0.dtd">
<facelet-taglib>
  <namespace>http://tutorialspoint.com/facelets</namespace>
  <tag>
    <tag-name>buttonPanel</tag-name>
    <source>com/tutorialspoint/buttonPanel.xhtml</source>
  </tag>
</facelet-taglib>

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>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <context-param>
    <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
    <param-value>/WEB-INF/tutorialspoint.taglib.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
</web-app>
<xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets"
     xmlns:tp="http://tutorialspoint.com/facelets">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h1>Custom Tags Example</h1>
    <tp:buttonPanel okLabel="Ok" cancelLabel="Cancel"/>
  </h:body>
</html>

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

**ui:remove Tag**

ui:remove tag is used to prevent the JSF specific code to be rendered on the client side. It is used especially to prevent commented out code to be rendered on the client side.

**JSF Tag Commented Out Using HTML Comment**

```
<!-- JSF code commented out -->

<!--
<h:commandButton value="Ok" />
-->
```

**Rendered Output**
Now using remove tag we'll see the following change in rendered output.

**JSF Tag Commented Out Using Remove Tag**

```xml
<!-- JSF code commented out -->
<ui:remove>
  <h:commandButton value="Ok" />
</ui:remove>
```

**Rendered Output**

```xml
<!-- JSF code commented out -->
```

**Example Application**

Let us create a test JSF application to test the template tags in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
</tbody>
</table>
Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <ui:remove>
      <h:commandButton value="Ok" />
    </ui:remove>
  <!--
    <h:commandButton value="Cancel" />
    -->
    </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, you'll see an empty page.

View source of the page and you will see the following html text.

**home.jsf**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <head>
    <title>JSF tutorial</title>
  </head>
  <body>
```

212
<!--
    &lt;h:commandButton value=&quot;Cancel&quot; /&gt;
-->
</body>
</html>
JSF provides inbuilt convertors to convert its UI component's data to object used in a managed bean and vice versa. For example, these tags can convert a text into date object and can validate the format of input as well.

For these tags, you need to use the following namespaces of URI in html node.

```html
<html
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
>
```

Following are important Convertor Tags in JSF 2.0:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f:convertNumber</td>
</tr>
<tr>
<td></td>
<td>Converts a String into a Number of desired format</td>
</tr>
<tr>
<td>2</td>
<td>f:convertDateTime</td>
</tr>
<tr>
<td>3</td>
<td>Custom Convertor</td>
</tr>
<tr>
<td></td>
<td>Creating a custom convertor</td>
</tr>
</tbody>
</table>

**f:convertNumber**

f:convertNumber tag is used to convert a string value to a number of required format.

**JSF Tag**

```html
<f:convertNumber minFractionDigits="2" />  
```
<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>type</strong>&lt;br&gt;number (default), currency, or percent</td>
</tr>
<tr>
<td>2</td>
<td><strong>pattern</strong>&lt;br&gt;Formatting pattern, as defined in java.text.DecimalFormat</td>
</tr>
<tr>
<td>3</td>
<td><strong>maxFractionDigits</strong>&lt;br&gt;Maximum number of digits in the fractional part</td>
</tr>
<tr>
<td>4</td>
<td><strong>minFractionDigits</strong>&lt;br&gt;Minimum number of digits in the fractional part</td>
</tr>
<tr>
<td>5</td>
<td><strong>maxIntegerDigits</strong>&lt;br&gt;Maximum number of digits in the integer part</td>
</tr>
<tr>
<td>6</td>
<td><strong>minIntegerDigits</strong>&lt;br&gt;Minimum number of digits in the integer part</td>
</tr>
<tr>
<td>7</td>
<td><strong>integerOnly</strong>&lt;br&gt;True, if only the integer part is parsed (default: false)</td>
</tr>
<tr>
<td>8</td>
<td><strong>groupingUsed</strong>&lt;br&gt;True, if grouping separators are used (default: true)</td>
</tr>
<tr>
<td>9</td>
<td><strong>locale</strong>&lt;br&gt;Locale whose preferences are to be used for parsing and formatting</td>
</tr>
<tr>
<td>10</td>
<td><strong>currencyCode</strong>&lt;br&gt;ISO 4217 currency code to use when converting currency values</td>
</tr>
<tr>
<td>11</td>
<td><strong>currencySymbol</strong>&lt;br&gt;Currency symbol to use when converting currency values</td>
</tr>
</tbody>
</table>
**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**home.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>ConvertNumber Example</h2>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
f:convertDateTime

f:convertDateTime tag is used to convert a string value to a date of required format. It also acts as a validator, a required date format.

JSF Tag

```xml
<f:convertDateTime pattern="dd-mm-yyyy" />
```

Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>date (default), time, or both</td>
</tr>
<tr>
<td>2</td>
<td>dateStyle</td>
</tr>
<tr>
<td></td>
<td>default, short, medium, long, or full</td>
</tr>
<tr>
<td>3</td>
<td>timeStyle</td>
</tr>
<tr>
<td></td>
<td>default, short, medium, long, or full</td>
</tr>
<tr>
<td>Step</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>Create a project with a name <strong>helloworld</strong> under a package <strong>com.tutorialspoint.test</strong> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <strong>home.xhtml</strong> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <strong>result.xhtml</strong> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <strong>UserData.java</strong> as a managed bean under package <strong>com.tutorialspoint.test</strong> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure business logic is working as per the requirements.</td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>ConvertDateTime Example</h2>
    <h:form>
      <h:inputText id="dateInput" value="#{userData.date}" label="Date">
        <f:convertDateTime pattern="dd-mm-yyyy"/>
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
    </h:form>
    <br/>
    <h:message for="dateInput" style="color:red"/>
    <h:outputText value="12-01-2012">
      <f:convertDateTime pattern="dd-mm-yyyy"/>
    </h:outputText>
  </h:body>
</html>
```

result.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:body>
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter any invalid value and press Submit button. See the following error message.
Enter any valid value and press Submit button. See the following result.

![Result]

**Custom Converter**

We can create our own Custom convertor in JSF.

Defining a custom converter in JSF is a three-step process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a converter class by implementing <code>javax.faces.convert.Converter</code> interface.</td>
</tr>
<tr>
<td>2</td>
<td>Implement <code>getAsObject()</code> and <code>getAsString()</code> methods of above interface.</td>
</tr>
<tr>
<td>3</td>
<td>Use Annotation <code>@FacesConverter</code> to assign a unique id to the custom convertor.</td>
</tr>
</tbody>
</table>

**Step 1: Create a Converter Class : UrlConverter.java**

```java
public class UrlConverter implements Converter {
    ...
}
```
Step 2: Implement Converter Interface Methods: UrlConverter.java

Create a simple class to store data: UrlData. This class will store a URL string.

```java
public class UrlData {

    private String url;

    public UrlData(String url){
        this.url = url;
    }

    ...
}
```

Use UrlData in getAsObject method.

```java
public class UrlConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext facesContext, UIComponent component, String value) {
        ...
        UrlData urlData = new UrlData(value.toString());
        return urlData;
    }

    @Override
    public String getAsString(FacesContext facesContext, UIComponent component, Object value) {
        return value.toString();
    }
}
```

Step 3: Annotate to Register the Converter: UrlConverter.java

```java
@FacesConverter("com.tutorialspoint.test.UrlConverter")
public class UrlConverter implements Converter {
}
```
Use the Convertor in JSF Page

```xml
<h:inputText id="urlInput" value="#{userData.data}" label="URL" />
   <f:converter converterId="com.tutorialspoint.test.UrlConverter" />
</h:inputText>
```

Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create <em>UrlData.java</em> under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>UrlConvertor.java</em> as a converter under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <em>UserData.java</em> as a managed bean under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep rest of the files unchanged.</td>
</tr>
<tr>
<td>6</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
</tr>
</tbody>
</table>
7. Compile and run the application to make sure the business logic is working as per the requirements.

8. Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

9. Launch your web application using appropriate URL as explained below in the last step.

**UrlData.java**

```java
package com.tutorialspoint.test;

public class UrlData {
    private String url;

    public UrlData(String url) {
        this.url = url;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String toString() {
        return url;
    }
}
```
class UrlConverter implements Converter {
  @Override
  public Object getAsObject(FacesContext facesContext, UIComponent component, String value) {
    StringBuilder url = new StringBuilder();

    if(!value.startsWith("http://", 0)) {
      url.append("http://");
    }
    url.append(value);

    try {
      new URI(url.toString());
    } catch (URISyntaxException e) {
      FacesMessage msg = new FacesMessage("Error converting URL", "Invalid URL format");
      msg.setSeverity(FacesMessage.SEVERITY_ERROR);
      throw new ConverterException(msg);
    }

    UrlData urlData = new UrlData(url.toString());
    return urlData;
  }
}
@Override
public String getAsString(FacesContext facesContext,
   UIComponent component, Object value) {
   return value.toString();
}

UserData.java

package com.tutorialspoint.test;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

private static final long serialVersionUID = 1L;

public UrlData data;

public UrlData getData() {
   return data;
}

public void setData(UrlData data) {
   this.data = data;
}
}
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
 <h:head>
  <title>JSF tutorial</title>
 </h:head>
 <h:body>
  <h2>Custom Converter Example</h2>
  <h:form>
   <h:inputText id="urlInput" value="#{userData.data}"
                label="URL" />
   <f:converter converterId="com.tutorialspoint.test.UrlConverter" />
  </h:form>
  <h:commandButton value="submit" action="result" />
  <h:message for="urlInput" style="color:red" />
 </h:body>
</html>
```

result.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:ui="http://java.sun.com/jsf/facelets">
 <h:body>
  <h2>Result</h2>
  <hr />
  #{userData.data}
 </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter any invalid value and press Submit button. See the following error message.

Enter any valid value and press Submit button. See the following result.
JSF provides inbuilt validators to validate its UI components. These tags can validate the length of the field, the type of input which can be a custom object.

For these tags you need to use the following namespaces of URI in html node.

```html
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core"
>
</html>
```

Following are important *Validator Tags* in JSF 2.0:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>f:validateLength</code> Validates the length of a string</td>
</tr>
<tr>
<td>2</td>
<td><code>f:validateLongRange</code> Validates the range of a numeric value</td>
</tr>
<tr>
<td>3</td>
<td><code>f:validateDoubleRange</code> Validates the range of a float value</td>
</tr>
<tr>
<td>4</td>
<td><code>f:validateRegex</code> Validates JSF component with a given regular expression</td>
</tr>
<tr>
<td>5</td>
<td>Custom Validator Creates a custom validator</td>
</tr>
</tbody>
</table>
**f:validateLength**

*f:validateLength* tag is used to validate the length of a string value in a particular range.

**JSF Tag**

```xml
<f:validateLength minimum="5" maximum="8" />
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>minimum A String with a minimum number of characters</td>
</tr>
<tr>
<td>2</td>
<td>maximum A String with a maximum number of characters</td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
</tr>
</tbody>
</table>
Create `UserData.java` as a managed bean under package `com.tutorialspoint.test` as explained below.

Compile and run the application to make sure the business logic is working as per the requirements.

Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;
    
    public String getName() {
        return name;
    }
    
    public void setName(String name) {
        this.name = name;
    }
}
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h:form>
            <h:inputText id="nameInput" value="#{userData.name}" label="name">
                <f:validateLength minimum="5" maximum="8" />
            </h:inputText>
            <h:commandButton value="submit" action="result" />
            <h:message for="nameInput" style="color:red" />
        </h:form>
    </h:body>
</html>
```

result.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
    </h:head>
    <h:body>
        <h:form>
            <h:inputText id="nameInput" value="#{userData.name}" label="name">
                <f:validateLength minimum="5" maximum="8" />
            </h:inputText>
            <h:commandButton value="submit" action="result" />
            <h:message for="nameInput" style="color:red" />
        </h:form>
    </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter an invalid value. Following will be the output.

Enter a valid value. Following will be the output.
**f:validateLongRange**

`f:validateLongRange` tag is used to validate the long value in a particular range.

**JSF Tag**

```xml
<f:validateLongRange minimum="5" maximum="200" />
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>minimum</code> Minimum long value within an optional range</td>
</tr>
<tr>
<td>2</td>
<td><code>maximum</code> Maximum long value within an optional range</td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>result.xhtml</code> in the webapps directory as explained below.</td>
</tr>
</tbody>
</table>
4. Create `UserData.java` as a managed bean under package `com.tutorialspoint.test` as explained below.

5. Compile and run the application to make sure the business logic is working as per the requirements.

6. Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

7. Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private int age;

    public int getAge() {
        return age;
    }

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

```html
<?xml version="1.0" encoding="UTF-8"?>
<!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"
xmls:h="http://java.sun.com/jsf/html"
xmls:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h:form>
      <h:inputText id="ageInput" value="#{userData.age}"
        label="age" />
      <f:validateLongRange minimum="5" maximum="200" />
      <h:commandButton value="submit" action="result" />
      <h:message for="ageInput" style="color:red" />
    </h:form>
  </h:body>
</html>
```

result.xhtml

```html
<?xml version="1.0" encoding="UTF-8"?>
<!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"
xmls:f="http://java.sun.com/jsf/core"
xmls:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    Age:  #{userData.age}
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter an invalid value. Following will be the output.

Enter a valid value. Following will be the output.

**f:validateDoubleRange**

f:validateDoubleRange tag is used to validate a value to a range of float values.

**JSF Tag**

```html
<f:validateDoubleRange minimum="1000.50" maximum="10000.50" />
```
Tag Attributes

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>minimum</td>
</tr>
<tr>
<td></td>
<td>Minimum double value within an optional range</td>
</tr>
<tr>
<td>2</td>
<td>maximum</td>
</tr>
<tr>
<td></td>
<td>Maximum double value within an optional range</td>
</tr>
</tbody>
</table>

Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>result.xhtml</em> in the webapps directory as explained below.</td>
</tr>
</tbody>
</table>
Create `UserData.java` as a managed bean under package `com.tutorialspoint.test` as explained below.

Compile and run the application to make sure the business logic is working as per the requirements.

Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private double salary;

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```
**home.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h:h:validateDoubleRange Example/>
    <h:form>
      <h:inputText id="salaryInput" value="#{userData.salary}"
                    label="salary">
        <f:validateDoubleRange minimum="1000.50" maximum="10000.50"/>
      </h:inputText>
      <h:commandButton value="submit" action="result"/>
      <h:message for="salaryInput" style="color:red"/>
    </h:form>
  </h:body>
</html>
```

**result.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>JSF Tutorial!</title>
  </h:head>
  <h:body>
    <h2>Result</h2>
    Salary: #{userData.salary}
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter an invalid value. Following will be the output.

Enter a valid value. Following will be the output.

**f:validateRegex**

f:validateRegex tag is used to validate a string value to a required format.

**JSF Tag**

```xml
<f:validateRegex pattern="((?=.*[a-z]).{6,})" />
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pattern</td>
</tr>
<tr>
<td></td>
<td>Formatting pattern</td>
</tr>
</tbody>
</table>
Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>result.xhtml</code> in the webapps directory as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>UserData.java</code> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String password;

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

**home.xhtml**

```xml
<!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">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>h:validateRegex Example</h2>
        <!-- password contains lower case letters only and.
        length of the password should be greater than 6. -->
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![JSF Tutorial Example](image-url)
Enter an invalid value. Following will be the output.

![Invalid value output](image)

Enter a valid value. Following will be the output.

![Valid value output](image)

## Custom Validator

We can create our own Custom validator in JSF.

Defining a custom validator in JSF is a three-step process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a validator class by implementing <code>javax.faces.validator.Validator</code> interface.</td>
</tr>
<tr>
<td>2</td>
<td>Implement <code>validate()</code> method of the above interface.</td>
</tr>
<tr>
<td>3</td>
<td>Use Annotation <code>@FacesValidator</code> to assign a unique ID to the custom validator.</td>
</tr>
</tbody>
</table>

**Step 1: Create a Validator Class : UrlValidator.java**

```java
public class UrlValidator implements Validator {
    ...
}
Step 2: Implement Validator Interface Methods : UrlValidator.java

```java
public class UrlValidator implements Validator {
    @Override
    public void validate(FacesContext facesContext,
                          UIComponent component,
                          String value) throws ValidatorException {
        ...
    }
}
```

Step 3: Annotate to Register the Validator : UrlValidator.java

```java
@FacesValidator("com.tutorialspoint.test.UrlValidator")
public class UrlValidator implements Validator {
}
```

Use the validator in JSF page

```html
<h:inputText id="urlInput" value="#{userData.data}" label="URL">
    <f:validator validatorId="com.tutorialspoint.test.UrlValidator" />
</h:inputText>
```

Example Application

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create UrlValidator.java as a converter under package com.tutorialspoint.test as explained below.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>UserData.java</em> as a managed bean under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>5</td>
<td>Create <code>result.xhtml</code> in the <code>webapps</code> directory as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

### `UrlValidator.java`

```java
package com.tutorialspoint.test;

import java.net.URI;
import java.net.URISyntaxException;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

@FacesValidator("com.tutorialspoint.test.UrlValidator")
public class UrlValidator implements Validator {

    @Override
```
public void validate(FacesContext facesContext, 
    UIComponent component, Object value) 
    throws ValidatorException {

    StringBuilder url = new StringBuilder();
    String urlValue = value.toString();

    if(!urlValue.startsWith("http://", 0)){
        url.append("http://");
    }
    url.append(urlValue);

    try {
        new URI(url.toString());
    } catch (URISyntaxException e) {
        FacesMessage msg = 
            new FacesMessage("URL validation failed","Invalid URL format");
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
        throw new ValidatorException(msg);
    }
}

UserData.java
package com.tutorialspoint.test;
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true) 
@SessionScoped 
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    public String data;
public String getData() {
    return data;
}

public void setData(String data) {
    this.data = data;
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h:form>
            <h:inputText id="urlInput" value="#{userData.data}" label="URL">
                <f:validator validatorId="com.tutorialspoint.test.UrlValidator" />
            </h:inputText>
            <h:commandButton value="submit" action="result" />
            <h:message for="urlInput" style="color:red" />
        </h:form>
    </h:body>
</html>

result.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter any invalid value and press Submit button. See the following error message.

Enter any valid value and press Submit button. Following will be the output.
JSF provides a rich control named DataTable to render and format HTML tables.

- DataTable can iterate over a collection or array of values to display data.
- DataTable provides attributes to modify its data in an easy way.

**HTML Header**

```html
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html">
</html>
```

Following are important *DataTable* operations in JSF 2.0:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Display DataTable</strong></td>
</tr>
<tr>
<td></td>
<td>How to display a dataTable</td>
</tr>
<tr>
<td>2</td>
<td><strong>Add data</strong></td>
</tr>
<tr>
<td></td>
<td>How to add a new row in a dataTable</td>
</tr>
<tr>
<td>3</td>
<td><strong>Edit data</strong></td>
</tr>
<tr>
<td></td>
<td>How to edit a row in a dataTable</td>
</tr>
<tr>
<td>4</td>
<td><strong>Delete data</strong></td>
</tr>
<tr>
<td></td>
<td>How to delete a row in dataTable</td>
</tr>
<tr>
<td>5</td>
<td><strong>Using DataModel</strong></td>
</tr>
<tr>
<td></td>
<td>Use DataModel to display row numbers in a dataTable</td>
</tr>
</tbody>
</table>
Display DataTable

*h: dataTable* tag is used to display data in a tabular fashion.

**JSF Tag**

```xml
<h:dataTable value="#{userData.employees}" var="employee"
    styleClass="employeeTable"
    headerClass="employeeTableHeader"
    rowClasses="employeeTableOddRow,employeeTableEvenRow">
    <h:column>
        <f:facet name="header">Name</f:facet>
        #{employee.name}
    </h:column>
    <h:column>
        <f:facet name="header">Department</f:facet>
        #{employee.department}
    </h:column>
    <h:column>
        <f:facet name="header">Age</f:facet>
        #{employee.age}
    </h:column>
    <h:column>
        <f:facet name="header">Salary</f:facet>
        #{employee.salary}
    </h:column>
</h:dataTable>
```

**Rendered Output**

```html
<table class="employeeTable">
    <thead>
        <tr>
            <th class="employeeTableHeader" scope="col">Name</th>
            <th class="employeeTableHeader" scope="col">Department</th>
            <th class="employeeTableHeader" scope="col">Age</th>
            <th class="employeeTableHeader" scope="col">Salary</th>
        </tr>
    </thead>
    <tbody>
        <!-- Render the data here -->
    </tbody>
</table>
```
<table>
  <tr>
    <td>John</td>
    <td>Marketing</td>
    <td>30</td>
    <td>2000.0</td>
  </tr>
  <tr class="employeeTableEvenRow">
    <td>Robert</td>
    <td>Marketing</td>
    <td>35</td>
    <td>3000.0</td>
  </tr>
</table>

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>Identifier for a component</td>
</tr>
<tr>
<td>2</td>
<td>rendered</td>
</tr>
<tr>
<td></td>
<td>A boolean; false suppresses rendering</td>
</tr>
<tr>
<td>3</td>
<td>dir</td>
</tr>
<tr>
<td></td>
<td>Direction for text. Valid values are ltr (left to right) and rtl (right to left)</td>
</tr>
<tr>
<td>4</td>
<td>styleClass</td>
</tr>
<tr>
<td></td>
<td>Cascading stylesheet (CSS) class name</td>
</tr>
<tr>
<td>5</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>A component's value, typically a value binding</td>
</tr>
<tr>
<td></td>
<td>Property</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------</td>
</tr>
<tr>
<td>6</td>
<td>bgcolor</td>
</tr>
<tr>
<td>7</td>
<td>border</td>
</tr>
<tr>
<td>8</td>
<td>cellspacing</td>
</tr>
<tr>
<td>9</td>
<td>cellspacing</td>
</tr>
<tr>
<td>10</td>
<td>columnClasses</td>
</tr>
<tr>
<td>11</td>
<td>first</td>
</tr>
<tr>
<td>12</td>
<td>footerClass</td>
</tr>
<tr>
<td>13</td>
<td>frame</td>
</tr>
<tr>
<td>14</td>
<td>headerClass</td>
</tr>
<tr>
<td>15</td>
<td>rowClasses</td>
</tr>
<tr>
<td>16</td>
<td>rules</td>
</tr>
<tr>
<td>17</td>
<td>summary</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| 18 | **var**  
The name of the variable created by the data table that represents the current item in the value |
| 19 | **title**  
A title, used for accessibility, that describes an element. Visual browsers typically create tooltips for the title’s value |
| 20 | **width**  
Width of an element |
| 21 | **onblur**  
Element loses focus |
| 22 | **onchange**  
Element’s value changes |
| 23 | **onclick**  
Mouse button is clicked over the element |
| 24 | **ondblclick**  
Mouse button is double-clicked over the element |
| 25 | **onfocus**  
Element receives focus |
| 26 | **onkeydown**  
Key is pressed |
<table>
<thead>
<tr>
<th></th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>onkeypress</td>
<td>Key is pressed and subsequently released</td>
</tr>
<tr>
<td>28</td>
<td>onkeyup</td>
<td>Key is released</td>
</tr>
<tr>
<td>29</td>
<td>onmousedown</td>
<td>Mouse button is pressed over the element</td>
</tr>
<tr>
<td>30</td>
<td>onmousemove</td>
<td>Mouse moves over the element</td>
</tr>
<tr>
<td>31</td>
<td>onmouseout</td>
<td>Mouse leaves the element’s area</td>
</tr>
<tr>
<td>32</td>
<td>onmouseover</td>
<td>Mouse moves onto an element</td>
</tr>
<tr>
<td>33</td>
<td>onmouseup</td>
<td>Mouse button is released</td>
</tr>
</tbody>
</table>

**Example Application**

Let us create a test JSF application to test the above tag.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - h:outputStylesheet</em> sub-chapter of <em>JSF - Basic Tags</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>styles.css</em> as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>Employee.java</em> under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Create <em>UserData.java</em> as a managed bean under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webs server.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**styles.css**

```css
.employeeTable{
    border-collapse:collapse;
    border:1px solid #000000;
}

.employeeTableHeader{
    text-align:center;
    background:none repeat scroll 0 0 #B5B5B5;
    border-bottom:1px solid #000000;
    padding:2px;
}

.employeeTableOddRow{
    text-align:center;
    background:none repeat scroll 0 0 #FFFFFF;
}

.employeeTableEvenRow{
    text-align:center;
    background:none repeat scroll 0 0 #D3D3D3;
}
```

**Employee.java**

```java
package com.tutorialspoint.test;

public class Employee {
    private String name;
    private String department;
```
private int age;
private double salary;
private boolean canEdit;

public Employee (String name, String department, int age, double salary) {
    this.name = name;
    this.department = department;
    this.age = age;
    this.salary = salary;
    canEdit = false;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDepartment() {
    return department;
}

public void setDepartment(String department) {
    this.department = department;
}

public int getAge() {
    return age;
}

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

public double getSalary() {
    return salary;
}
public void setSalary(double salary) {
    this.salary = salary;
}

public boolean isCanEdit() {
    return canEdit;
}

public void setCanEdit(boolean canEdit) {
    this.canEdit = canEdit;
}
```java
= new ArrayList<Employee>(Arrays.asList(
    new Employee("John", "Marketing", 30, 2000.00),
    new Employee("Robert", "Marketing", 35, 3000.00),
    new Employee("Mark", "Sales", 25, 2500.00),
    new Employee("Chris", "Marketing", 33, 2500.00),
    new Employee("Peter", "Customer Care", 20, 1500.00)
));

public ArrayList<Employee> getEmployees() {
    return employees;
}

public String addEmployee() {
    Employee employee = new Employee(name, dept, age, salary);
    employees.add(employee);
    return null;
}

public String deleteEmployee(Employee employee) {
    employees.remove(employee);
    return null;
}

public String editEmployee(Employee employee){
    employee.setCanEdit(true);
    return null;
}

public String saveEmployees(){
    //set "canEdit" of all employees to false
    for (Employee employee : employees){
        employee.setCanEdit(false);
    }
    return null;
}

public String getName() {
```
return name;
}

public void setName(String name) {
    this.name = name;
}

public String getDepartment() {
    return department;
}

public void setDepartment(String department) {
    this.department = department;
}

public int getAge() {
    return age;
}

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

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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"
  <h:head>
    <title>JSF tutorial</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>
  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h:dataTable value="#{userData.employees}" var="employee"
        styleClass="employeeTable"
        headerClass="employeeTableHeader"
        rowClasses="employeeTableOddRow,employeeTableEvenRow">
        <h:column>
          <f:facet name="header">Name</f:facet>
          #{employee.name}
        </h:column>
        <h:column>
          <f:facet name="header">Department</f:facet>
          #{employee.department}
        </h:column>
        <h:column>
          <f:facet name="header">Age</f:facet>
          #{employee.age}
        </h:column>
        <h:column>
          <f:facet name="header">Salary</f:facet>
          #{employee.salary}
        </h:column>
      </h:dataTable>
    </h:form>
  </h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![DataTable Example](image)

### Add Data to DataTable

In this section, we'll showcase adding a row to a `DataTable`.

### Example Application

Let us create a test JSF application to test the above functionality.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the <a href="https://example.com">JSF - Display DataTable sub-chapter</a> of <a href="https://example.com">JSF - DataTables chapter</a>.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
</tbody>
</table>
Launch your web application using appropriate URL as explained below in the last step.

**home.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:head>
    <title>JSF tutorial</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>
  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h:dataTable value="#{userData.employees}" var="employee"
                   styleClass="employeeTable"
                   headerClass="employeeTableHeader"
                   rowClasses="employeeTableOddRow,employeeTableEvenRow">
        <h:column>
          <f:facet name="header">Name</f:facet>
          #{employee.name}
        </h:column>
        <h:column>
          <f:facet name="header">Department</f:facet>
          #{employee.department}
        </h:column>
        <h:column>
          <f:facet name="header">Age</f:facet>
          #{employee.age}
        </h:column>
        <h:column>
          <f:facet name="header">Salary</f:facet>
        </h:column>
      </h:dataTable>
    </h:form>
  </h:body>
</html>
```
#{employee.salary}
</h:column>
</h:dataTable>
<h3>Add Employee</h3>
<hr/>
<table>
<tr>
<td>Name : </td>
<td><h:inputText size="10" value="#{userData.name}" /></td>
</tr>
<tr>
<td>Department : </td>
<td><h:inputText size="20" value="#{userData.dept}" /></td>
</tr>
<tr>
<td>Age : </td>
<td><h:inputText size="5" value="#{userData.age}" /></td>
</tr>
<tr>
<td>Salary : </td>
<td><h:inputText size="5" value="#{userData.salary}" /></td>
</tr>
</table>
<h:form>
<h:commandButton value="Add Employee"
action="#{userData.addEmployee}" />
</h:form>
</h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Add values to Add Employee Form and click Add Employee button. See the following result.
Edit Data of a DataTable

In this section, we'll showcase the adding editing capability to a row in a dataTable.

Example Application

Let us create a test JSF application to test the above functionality.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - Display DataTable sub-chapter of JSF - Data Tables chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify home.xhtml as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <h:head>
        <title>JSF tutorial</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h2>DataTable Example</h2>
        <h:form>
            <h:dataTable value="#{userData.employees}" var="employee" styleClass="employeeTable"
                          headerClass="employeeTableHeader"
                          rowClasses="employeeTableOddRow,employeeTableEvenRow">
                <h:column>
<f:facet name="header">Name</f:facet>
<h:inputText value="#{employee.name}"
size="10" rendered="#{employee.canEdit}" />
<h:outputText value="#{employee.name}"
rendered="#{not employee.canEdit}" />
</h:column>

<f:facet name="header">Department</f:facet>
<h:inputText value="#{employee.department}"
size="20" rendered="#{employee.canEdit}" />
<h:outputText value="#{employee.department}"
rendered="#{not employee.canEdit}" />
</h:column>

<f:facet name="header">Age</f:facet>
<h:inputText value="#{employee.age}" size="5"
rendered="#{employee.canEdit}" />
<h:outputText value="#{employee.age}"
rendered="#{not employee.canEdit}" />
</h:column>

<f:facet name="header">Salary</f:facet>
<h:inputText value="#{employee.salary}" size="5"
rendered="#{employee.canEdit}" />
<h:outputText value="#{employee.salary}"
rendered="#{not employee.canEdit}" />
</h:column>

<f:facet name="header">Edit</f:facet>
<h:commandButton value="Edit"
action="#{userData.editEmployee}"
rendered="#{not employee.canEdit}">
<f:setPropertyActionListener
    target="#{userData.employee}" value="#{employee}" />
</h:commandButton>
</h:column>

</h:dataTable>
<br/>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Click the edit button of any row. Following will be the output.
Click Save Employees button to save the edit. Following will be the output

Delete Data of a DataTable

In this section, we’ll showcase the adding deleting capability in dataTable.

Example Application

Let us create a test JSF application to test the above functionality.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - Display DataTable sub-chapter of JSF - DataTables chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify home.xhtml as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>3</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>4</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>5</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?
<!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">
  <h:head>
    <title>JSF tutorial</title>
    <h:outputStylesheet library="css" name="styles.css" />
  </h:head>
  <h:body>
    <h2>DataTable Example</h2>
    <h:form>
      <h: dataTable value="#{userData.employees}" var="employee"
                    styleClass="employeeTable"
                    headerClass="employeeTableHeader"
                    rowClasses="employeeTableOddRow,employeeTableEvenRow">
        <h:column>
          <f:facet name="header">Name</f:facet>
          <h:inputText value="#{employee.name}"
                        size="10" rendered="#{employee.canEdit}" />
          <h:outputText value="#{employee.name}"
                         rendered="#{not employee.canEdit}" />
        </h:column>
        <h:column>
          <f:facet name="header">Department</f:facet>
          <h:inputText value="#{employee.department}"
                        size="20" rendered="#{employee.canEdit}" />
          <h:outputText value="#{employee.department}"
                         rendered="#{not employee.canEdit}" />
        </h:column>
        <h:column>
          <f:facet name="header">Age</f:facet>
          <h:inputText value="#{employee.age}" size="5"
                        rendered="#{employee.canEdit}" />
          <h:outputText value="#{employee.age}"
                         rendered="#{not employee.canEdit}" />
        </h:column>
      </h: dataTable>
    </h:form>
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![DataTable Example](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Age</th>
<th>Salary</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Marketing</td>
<td>30</td>
<td>2000.0</td>
<td>Delete</td>
</tr>
<tr>
<td>Robert</td>
<td>Marketing</td>
<td>35</td>
<td>3500.0</td>
<td>Delete</td>
</tr>
<tr>
<td>Mark</td>
<td>Sales</td>
<td>25</td>
<td>2500.0</td>
<td>Delete</td>
</tr>
<tr>
<td>Chris</td>
<td>Marketing</td>
<td>33</td>
<td>2500.0</td>
<td>Delete</td>
</tr>
<tr>
<td>Peter</td>
<td>Customer Care</td>
<td>20</td>
<td>1500.0</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Click *delete* button of any row. Following will be the output.

![DataTable Example](image)

**Using DataModel in a DataTable**

In this section, we'll showcase the use of datamodel in a dataTable.

**Example Application**

Let us create a test JSF application to test the above functionality.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - Display DataTable</em> sub-chapter of <em>JSF - DataTables</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>UserData.java</em> as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Modify <em>home.xhtml</em> as explained below. Keep the rest of the files unchanged.</td>
</tr>
</tbody>
</table>
Compile and run the application to make sure the business logic is working as per the requirements.

Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.model.ArrayDataModel;
import javax.faces.model.DataModel;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private static final Employee[] employees = new Employee[] {
        new Employee("John", "Marketing", 30, 2000.00),
        new Employee("Robert", "Marketing", 35, 3000.00),
        new Employee("Mark", "Sales", 25, 2500.00),
        new Employee("Chris", "Marketing", 33, 2500.00),
        new Employee("Peter", "Customer Care", 20, 1500.00)
    };

    private DataModel<Employee> employeeDataModel
```
= new ArrayDataModel<Employee>(employees);

public DataModel<Employee> getEmployees() {
    return employeeDataModel;
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <h:head>
        <title>JSF tutorial</title>
        <h:outputStylesheet library="css" name="styles.css" />
    </h:head>
    <h:body>
        <h: dataTable value="#{userData.employees}" var="employee"
                      styleClass="employeeTable"
                      headerClass="employeeTableHeader"
                      rowClasses="employeeTableOddRow,employeeTableEvenRow">
            <h:column>
                <f:facet name="header">Sr. No</f:facet>
                #{userData.employees.RowIndex + 1}
            </h:column>
            <h:column>
                <f:facet name="header">Name</f:facet>
                #{employee.name}
            </h:column>
            <h:column>
                <f:facet name="header">Department</f:facet>
                #{employee.department}
            </h:column>
        </h: dataTable>
    </h:body>
</html>
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
JSF provides the developers with a powerful capability to define their own custom components, which can be used to render custom contents.

**Define Custom Component**

Defining a custom component in JSF is a two-step process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Create a resources folder. Create a xhtml file in resources folder with a composite namespace.</td>
</tr>
<tr>
<td>1b</td>
<td>Use composite tags <code>composite:interface</code>, <code>composite:attribute</code> and <code>composite:implementation</code>, to define content of the composite component. Use <code>cc.attrs</code> in <code>composite:implementation</code> to get variable defined using <code>composite:attribute</code> in <code>composite:interface</code>.</td>
</tr>
</tbody>
</table>

### Step 1a: Create Custom Component: loginComponent.xhtml

Create a folder tutorialspoint in resources folder and create a file loginComponent.xhtml in it.

Use composite namespace in html header.

```html
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:f="http://java.sun.com/jsf/core"
     xmlns:composite="http://java.sun.com/jsf/composite" >
...
</html>
```
Step 1b: Use Composite Tags : loginComponent.xhtml

Following table describes the use of composite tags.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Tag &amp; Description</th>
</tr>
</thead>
</table>
| 1       | composite:interface  
          Declare configurable values to be used in composite:implementation |
| 2       | composite:attribute  
          Configuration values are declared using this tag |
| 3       | composite:implementation  
          Declares JSF component. Can access the configurable values defined in composite:interface using #{cc.attrs.attribute-name} expression |

```xml
<composite:interface>
    <composite:attribute name="usernameLabel" />
    <composite:attribute name="usernameValue" />
</composite:interface>

<composite:implementation>
    <h:form>
        #{cc.attrs.usernameLabel} :
        <h:inputText id="username" value="#{cc.attrs.usernameValue}" />
    </h:form>
</composite:implementation>
```
Use Custom Component

Using a custom component in JSF is a simple process.

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2a</td>
<td>Create a xhtml file and use custom component's namespace. Namespace will be <a href="http://java.sun.com/jsf/">http://java.sun.com/jsf/</a>&lt;folder-name&gt; where folder-name is folder in resources directory containing the custom component</td>
</tr>
<tr>
<td>2b</td>
<td>Use the custom component as normal JSF tags</td>
</tr>
</tbody>
</table>

Step 2a: Use Custom Namespace: home.xhtml

```xml
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:ui="http://java.sun.com/jsf.facelets"
     xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">

Step 2b: Use Custom Tag: home.xhtml and Pass Values

```xml
<h:form>
    <tp:loginComponent
        usernameLabel="Enter User Name: "
        usernameValue="#{userData.name}" />
</h:form>
```

Example Application

Let us create a test JSF application to test the custom component in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create resources folder under src -&gt; main folder.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>tutorialspoint</code> folder under <code>src -&gt; main -&gt; resources</code> folder.</td>
</tr>
<tr>
<td>4</td>
<td>Create <code>loginComponent.xhtml</code> file under <code>src -&gt; main -&gt; resources -&gt; tutorialspoint</code> folder.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <code>UserData.java</code> file as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>7</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>8</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>9</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

### `loginComponent.xhtml`

```xml
<composite:interface>
  <composite:attribute name="usernameLabel" />
  <composite:attribute name="usernameValue" />
  <composite:attribute name="passwordLabel" />
  <composite:attribute name="passwordValue" />
</composite:interface>
```
<html>
<composite:interface>
<composite:attribute name="loginButtonLabel" />
<composite:attribute name="loginButtonAction"
method-signature="java.lang.String login()" />
</composite:interface>
<composite:implementation>
<h:form>
<h:message for="loginPanel" style="color:red;" />
<h:panelGrid columns="2" id="loginPanel">
#{cc.attrs.usernameLabel} :
<h:inputText id="username" value="#{cc.attrs.usernameValue}" />
#{cc.attrs.passwordLabel} :
<h:inputSecret id="password" value="#{cc.attrs.passwordValue}" />
</h:panelGrid>
<h:commandButton action="#{cc.attrs.loginButtonAction}"
value="#{cc.attrs.loginButtonLabel}" />
</h:form>
</composite:implementation>
</html>

UserData.java

package com.tutorialspoint.test;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {

```java
public class UserData {
  private String name;
  private String password;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }

  public String login() {
    return "result";
  }
}

home.xhtml
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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"
  xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Custom Component Example</h2>
    <h:form>
      <tp:loginComponent
        usernameLabel="Enter User Name: ">
        usernameValue="#{userData.name}" 
        passwordLabel="Enter Password: ">
        passwordValue="#{userData.password}" 
        loginButtonLabel="Login" 
        loginButtonAction="#{userData.login}" />
    </h:form>
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
AJAX stands for Asynchronous JavaScript and XML.

Ajax is a technique to use HTTPXMLObject of JavaScript to send data to the server and receive data from the server asynchronously. Thus using Ajax technique, javascript code exchanges data with the server, updates parts of the web page without reloading the whole page.

JSF provides excellent support for making ajax call. It provides f:ajax tag to handle ajax calls.

**JSF Tag**

```html
<f:ajax execute="input-component-name" render="output-component-name" />
```

**Tag Attributes**

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Attribute &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>disabled</strong></td>
</tr>
<tr>
<td></td>
<td>If true, the Ajax behavior will be applied to any parent or child components. If false, the Ajax behavior will be disabled.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Event</strong></td>
</tr>
<tr>
<td></td>
<td>The event that will invoke Ajax requests, for example &quot;click&quot;, &quot;change&quot;, &quot;blur&quot;, &quot;keypress&quot;, etc.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Execute</strong></td>
</tr>
<tr>
<td></td>
<td>A space-separated list of IDs for components that should be included in the Ajax request.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Immediate</strong></td>
</tr>
<tr>
<td></td>
<td>If &quot;true&quot; behavior events generated from this behavior are broadcast during Apply Request Values phase. Otherwise, the events will be broadcast during Invoke Applications phase.</td>
</tr>
</tbody>
</table>
### Example Application

Let us create a test JSF application to test the custom component in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the <code>JSF - First Application</code> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>UserData.java</code> file as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
</tbody>
</table>
Compile and run the application to make sure the business logic is working as per the requirements.

Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String getWelcomeMessage(){
        return "Hello " + name;
    }
}
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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"
     xmlns:tp="http://java.sun.com/jsf/composite/tutorialspoint">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Ajax Example</h2>
    <h:form>
      <h:inputText id="inputName" value="#{userData.name}"/>
      <h:commandButton value="Show Message">
        <f:ajax execute="inputName" render="outputMessage" />
      </h:commandButton>
      <h:outputText id="outputMessage" value="#{userData.welcomeMessage !=null ? userData.welcomeMessage : ''}"
                     rendered="inputName"/>
    </h:form>
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Enter the name and press the *Show Message* button. You will see the following result without page refresh/form submit.
When a user clicks a JSF button or link or changes any value in the text field, JSF UI component fires an event, which will be handled by the application code. To handle such an event, an event handler is to be registered in the application code or managed bean.

When a UI component checks that a user event has occurred, it creates an instance of the corresponding event class and adds it to an event list. Then, Component fires the event, i.e., checks the list of listeners for that event and calls the event notification method on each listener or handler.

JSF also provide system level event handlers, which can be used to perform some tasks when the application starts or is stopping.

Following are some important Event Handler in JSF 2.0:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Event Handlers &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>valueChangeListener</td>
</tr>
<tr>
<td></td>
<td>Value change events get fired when the user make changes in input components.</td>
</tr>
<tr>
<td>2</td>
<td>actionListener</td>
</tr>
<tr>
<td></td>
<td>Action events get fired when the user clicks a button or link component.</td>
</tr>
<tr>
<td>3</td>
<td>Application Events</td>
</tr>
<tr>
<td></td>
<td>Events firing during JSF lifecycle: PostConstructApplicationEvent, PreDestroyApplicationEvent, PreRenderViewEvent.</td>
</tr>
</tbody>
</table>

**valueChangeListener**

When the user interacts with input components, such as h:inputText or h:selectOneMenu, the JSF fires a valueChangeEvent, which can be handled in two ways.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method Binding</td>
<td>Pass the name of the managed bean method in valueChangeListener attribute of UI Component.</td>
</tr>
<tr>
<td>ValueChangeListener</td>
<td>Implement ValueChangeListener interface and pass the implementation class name to valueChangeListener attribute of UI Component.</td>
</tr>
</tbody>
</table>
**Method Binding**

Define a method

```java
public void localeChanged(ValueChangeEvent e){
    //assign new value to country
    selectedCountry = e.getNewValue().toString();
}
```

Use the above method

```html
<h:selectOneMenu value="#{userData.selectedCountry}" onchange="submit()"
    valueChangeListener="#{userData.localeChanged}" >
    <f:selectItems value="#{userData.countries}" />
</h:selectOneMenu>
```

**ValueChangeListener**

Implement ValueChangeListener

```java
public class LocaleChangeListener implements ValueChangeListener {
    @Override
    public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
        //access country bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
            getExternalContext().getSessionMap().get("userData");
        userData.setSelectedCountry(event.getNewValue().toString());
    }
}
```

Use listener method

```html
<h:selectOneMenu value="#{userData.selectedCountry}" onchange="submit()">
    <f:valueChangeListener type="com.tutorialspoint.test.LocaleChangeListener" />
    <f:selectItems value="#{userData.countries}" />
</h:selectOneMenu>
```
## Example Application

Let us create a test JSF application to test the valueChangeListener in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>UserData.java</code> file as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>LocaleChangeListener.java</code> file under a package <code>com.tutorialspoint.test</code>. Modify it as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>5</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>6</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>7</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

### UserData.java

```java
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Map;
```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ValueChangeEvent;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

private static final long serialVersionUID = 1L;

private static Map<String,String> countryMap;
private String selectedCountry = "United Kingdom"; //default value

static{
    countryMap = new LinkedHashMap<String,String>();
    countryMap.put("en", "United Kingdom"); //locale, country name
    countryMap.put("fr", "French");
    countryMap.put("de", "German");
}

public void localeChanged(ValueChangeEvent e){
    //assign new value to country
    selectedCountry = e.getNewValue().toString();
}

public Map<String, String> getCountries() {
    return countryMap;
}

public String getSelectedCountry() {
    return selectedCountry;
}

public void setSelectedCountry(String selectedCountry) {
    this.selectedCountry = selectedCountry;
}
LocaleChangeListener.java

```java
package com.tutorialspoint.test;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;

public class LocaleChangeListener implements ValueChangeListener {
    @Override
    public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
        //access country bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
                getExternalContext().getSessionMap().get("userData");

        userData.setSelectedCountry(event.getNewValue().toString());
    }
}
```

home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>valueChangeListener Examples</h2>
        <h:form>
            <h2>Method Binding</h2>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Select locale. You will see the following result.

Modify **home.xhtml** again in the deployed directory where you've deployed the application as explained below. Keep the rest of the files unchanged.

**home.xhtml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>valueChangeListener Examples</h2>
    <h:form>
      <h2>ValueChangeListener interface</h2>
      <hr/>
      <h:panelGrid columns="2">
        Selected locale :
        <h:selectOneMenu value="#{userData.selectedCountry}"
                         onchange="submit()"
                         valueChangeListener="com.tutorialspoint.test.LocaleChangeListener" />
        <f:selectItems value="#{userData.countries}" />
      </h:panelGrid>
    </h:form>
  </h:body>
</html>
```
Once you are ready with all the changes done, refresh the page in the browser. If everything is fine with your application, this will produce the following result.

Select locale. You will see the following result.
**actionListener**

When the user interacts with the components, such as h:commandButton or h:link, the JSF fires action events which can be handled in two ways.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method Binding</td>
<td>Pass the name of the managed bean method in <code>actionListener</code> attribute of UI Component.</td>
</tr>
<tr>
<td>ActionListener</td>
<td>Implement ActionListener interface and pass the implementation class name to <code>actionListener</code> attribute of UI Component.</td>
</tr>
</tbody>
</table>

**Method Binding**

**Define a method**

```java
public void updateData(ActionEvent e){
    data="Hello World";
}
```

**Use the above method**

```xml
<h:commandButton id="submitButton"
    value="Submit" action="#{userData.showResult}"
    actionListener="#{userData.updateData}" />
</h:commandButton>
```

**ActionListener**

**Implement ActionListener**

```java
public class UserActionListener implements ActionListener{
    @Override
    public void processAction(ActionEvent arg0)
        throws AbortProcessingException {
        //access userData bean directly
        UserData userData = (UserData) FacesContext.getCurrentInstance().
            getExternalContext().getSessionMap().get("userData");
        userData.setData("Hello World");
    }
}
```
Use listener method

```xml
<h:commandButton id="submitButton1"
    value="Submit" action="#{userData.showResult}" />
<f:actionListener type="com.tutorialspoint.test.UserActionListener" />
</h:commandButton>
```

**Example Application**

Let us create a test JSF application to test the actionListener in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>UserData.java</code> file as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>UserActionListener.java</code> file under a package <code>com.tutorialspoint.test</code>. Modify it as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <code>result.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ValueChangeEvent;
```
@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

private static final long serialVersionUID = 1L;

    private static Map<String,String> countryMap;
    private String data = "sample data";

    public String showResult(){
        return "result";
    }

    public void updateData(ActionEvent e){
        data="Hello World";
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

UserActionListener.java

package com.tutorialspoint.test;

import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;

public class UserActionListener implements ActionListener{
    @Override
public void processAction(ActionEvent arg0) throws AbortProcessingException {
    // access userData bean directly
    UserData userData = (UserData) FacesContext.getCurrentInstance().
        getExternalContext().getSessionMap().get("userData");
    userData.setData("Hello World");
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
        <h2>actionListener Examples</h2>
        <h:form>
            <h2>Method Binding</h2>
            <hr/>
            <h:commandButton id="submitButton"
                value="Submit" action="#{userData.showResult}"
                actionListener="#{userData.updateData}" />
        </h:commandButton>
        <h:form>
            <h2>ActionListener interface</h2>
            <hr/>
            <h:commandButton id="submitButton1"
                value="Submit" action="#{userData.showResult}"
                type="com.tutorialspoint.test.UserActionListener" />
        </h:commandButton>
    </h:form>
</h:body>
</html>
result.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
      <title>JSF Tutorial!</title>
    </h:head>
    <h:body>
      <h2>Result</h2>
      <hr />
      #{userData.data}
    </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Click any submit button. You will see the following result.

**Application Events**

JSF provides system event listeners to perform application specific tasks during JSF application Life Cycle.

<table>
<thead>
<tr>
<th>System Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostConstructApplicationEvent</td>
<td>Fires when the application starts. Can be used to perform initialization tasks after the application has started.</td>
</tr>
<tr>
<td>PreDestroyApplicationEvent</td>
<td>Fires when the application is about to shut down. Can be used to perform cleanup tasks before the application is about to shut down.</td>
</tr>
<tr>
<td>PreRenderViewEvent</td>
<td>Fires before a JSF page is to be displayed. Can be used to authenticate the user and provide restricted access to JSF View.</td>
</tr>
</tbody>
</table>
System Events can be handled in the following manner.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemEventListener</td>
<td>Implement SystemEventListener interface and register the system-event-listener class in faces-config.xml</td>
</tr>
<tr>
<td>Method Binding</td>
<td>Pass the name of the managed bean method in listener attribute of f:event.</td>
</tr>
</tbody>
</table>

**SystemEventListener**

Implement SystemEventListener Interface.

```java
public class CustomSystemEventListener implements SystemEventListener {
    @Override
    public void processEvent(SystemEvent event) throws AbortProcessingException {
        if(event instanceof PostConstructApplicationEvent){
            System.out.println("Application Started. PostConstructApplicationEvent occurred!");
        }
    }
}
```

Register custom system event listener for system event in faces-config.xml.

```xml
<system-event-listener>
    <system-event-listener-class>
        com.tutorialspoint.test.CustomSystemEventListener
    </system-event-listener-class>
    <system-event-class>
        javax.faces.event.PostConstructApplicationEvent
    </system-event-class>
</system-event-listener>
```
Method Binding

Define a method.

```java
public void handleEvent(ComponentSystemEvent event){
    data="Hello World";
}
```

Use the above method.

```html
<f:event listener="#{user.handleEvent}" type="preRenderView" />
```

Example Application

Let us create a test JSF application to test the system events in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <code>helloworld</code> under a package <code>com.tutorialspoint.test</code> as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <code>UserData.java</code> file as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <code>CustomSystemEventListener.java</code> file under a package <code>com.tutorialspoint.test</code>. Modify it as explained below</td>
</tr>
<tr>
<td>4</td>
<td>Modify <code>home.xhtml</code> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Create <code>faces-config.xml</code> in <code>WEB-INF</code> folder. Modify it as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>6</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>7</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>8</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>
**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private String data = "sample data";

    public void handleEvent(ComponentSystemEvent event){
        data="Hello World";
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

**CustomSystemEventListener.java**

```java
package com.tutorialspoint.test;
import javax.faces.application.Application;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.PostConstructApplicationEvent;
```
import javax.faces.event.PreDestroyApplicationEvent;
import javax.faces.event.SystemEvent;
import javax.faces.event.SystemEventListener;

public class CustomSystemEventListener implements SystemEventListener {

    @Override
    public boolean isListenerForSource(Object value) {
        // only for Application
        return (value instanceof Application);
    }

    @Override
    public void processEvent(SystemEvent event)
        throws AbortProcessingException {
        if (event instanceof PostConstructApplicationEvent) {
            System.out.println("Application Started.
                PostConstructApplicationEvent occurred!");
        }
        if (event instanceof PreDestroyApplicationEvent) {
            System.out.println("PreDestroyApplicationEvent occurred.
                Application is stopping.");
        }
    }
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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">
    <h:head>
        <title>JSF tutorial</title>
    </h:head>
    <h:body>
<h2>Application Events Examples</h2>

```html
<f:event listener="#{userData.handleEvent}" type="preRenderView" />
#{userData.data}
</html>
```

**faces-config.xhtml**

```xml
<?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">
  <application>
    <!-- Application Startup -->
    <system-event-listener>
      <system-event-listener-class>
        com.tutorialspoint.test.CustomSystemEventListener
      </system-event-listener-class>
    </system-event-listener>
    <system-event-class>
      javax.faces.event.PostConstructApplicationEvent
    </system-event-class>
    </system-event-listener>
    <!-- Before Application is to shut down -->
    <system-event-listener>
      <system-event-listener-class>
        com.tutorialspoint.test.CustomSystemEventListener
      </system-event-listener-class>
      <system-event-class>
        javax.faces.event.PreDestroyApplicationEvent
      </system-event-class>
    </system-event-listener>
  </application>
</faces-config>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

Look into your web-server console output. You will see the following result.

```
INFO: Deploying web application archive helloworld.war
contextInitialized
INFO: Initializing Mojarra 2.1.7 (SNAPSHOT 20120206) for context '/helloworld'
Application Started. PostConstructApplicationEvent occurred!
Dec 6, 2012 8:21:46 AM com.sun.faces.config.ConfigureListener
$WebConfigResourceMonitor$Monitor <init>
INFO: Monitoring jndi:/localhost/helloworld/WEB-INF/faces-config.xml
for modifications
INFO: Starting Coyote HTTP/1.1 on http-8080
Dec 6, 2012 8:21:46 AM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009

Dec 6, 2012 8:21:46 AM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/24 config=null
Dec 6, 2012 8:21:46 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 44272 ms
```
In this article, we'll demonstrate how to integrate database in JSF using JDBC.

Following are the database requirements to run this example.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Software &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>PostgreSQL 9.1</strong></td>
</tr>
<tr>
<td></td>
<td>Open Source and lightweight database</td>
</tr>
<tr>
<td>2</td>
<td><strong>PostgreSQL JDBC4 Driver</strong></td>
</tr>
<tr>
<td></td>
<td>JDBC driver for PostgreSQL 9.1 and JDK 1.5 or above</td>
</tr>
</tbody>
</table>

Put PostgreSQL JDBC4 Driver jar in tomcat web server's lib directory.

**Database SQL Commands**

```sql
create user user1;

create database testdb with owner=user1;

CREATE TABLE IF NOT EXISTS authors (  
id int PRIMARY KEY,  
name VARCHAR(25)
);

INSERT INTO authors(id, name) VALUES(1, 'Rob Bal');  
INSERT INTO authors(id, name) VALUES(2, 'John Carter');  
INSERT INTO authors(id, name) VALUES(3, 'Chris London');  
INSERT INTO authors(id, name) VALUES(4, 'Truman De Bal');  
INSERT INTO authors(id, name) VALUES(5, 'Emile Capote');  
INSERT INTO authors(id, name) VALUES(7, 'Breech Jabber');  
INSERT INTO authors(id, name) VALUES(8, 'Bob Carter');  
INSERT INTO authors(id, name) VALUES(9, 'Nelson Mand');  
INSERT INTO authors(id, name) VALUES(10, 'Tennant Mark');
```
alter user user1 with password 'user1';

grant all on authors to user1;

**Example Application**
Let us create a test JSF application to test JDBC integration.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <strong>helloworld</strong> under a package <code>com.tutorialspoint.test</code> as explained in the <strong>JSF - First Application</strong> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create <strong>resources</strong> folder under <code>src -&gt; main</code> folder.</td>
</tr>
<tr>
<td>3</td>
<td>Create <strong>css</strong> folder under <code>src -&gt; main -&gt; resources</code> folder.</td>
</tr>
<tr>
<td>4</td>
<td>Create <strong>styles.css</strong> file under <code>src -&gt; main -&gt; resources -&gt; css</code> folder.</td>
</tr>
<tr>
<td>5</td>
<td>Modify <strong>styles.css</strong> file as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Modify <strong>pom.xml</strong> as explained below.</td>
</tr>
<tr>
<td>7</td>
<td>Create <strong>Author.java</strong> under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
</tbody>
</table>
8. Create `UserData.java` under package `com.tutorialspoint.test` as explained below.

9. Modify `home.xhtml` as explained below. Keep the rest of the files unchanged.

10. Compile and run the application to make sure the business logic is working as per the requirements.

11. Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

12. Launch your web application using appropriate URL as explained below in the last step.

---

**styles.css**

```css
.authorTable{
  border-collapse:collapse;
  border-bottom:1px solid #000000;
}

.authorTableHeader{
  text-align:center;
  background:none repeat scroll 0 0 #B5B5B5;
  border-bottom:1px solid #000000;
  border-top:1px solid #000000;
  padding:2px;
}

.authorTableOddRow{
  text-align:center;
  background:none repeat scroll 0 0 #FFFFFF;
}
```
pom.xml

```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.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld 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>2.1.7</version>
    </dependency>
    <dependency>
      <groupId>com.sun.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>2.1.7</version>
    </dependency>
  </dependencies>
</project>
```
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
</dependency>
<dependency>
<groupId>postgresql</groupId>
<artifactId>postgresql</artifactId>
<version>9.1-901.jdbc4</version>
</dependency>
</dependencies>
<build>
<finalName>helloworld</finalName>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.3.1</version>
<configuration>
<source>1.6</source>
<target>1.6</target>
</configuration>
</plugin>
<plugin>
<artifactId>maven-resources-plugin</artifactId>
<version>2.6</version>
<executions>
<execution>
<id>copy-resources</id>
<phase>validate</phase>
<goals>
<goal>copy-resources</goal>
</goals>
<configuration>
<outputDirectory>${basedir}/target/helloworld/resources
</outputDirectory>
<resources>
<resource>
<directory>src/main/resources</directory>
<filtering>true</filtering>
</resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>

**Author.java**

```java
package com.tutorialspoint.test;

public class Author {
    int id;
    String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```
UserData.java

```
package com.tutorialspoint.test;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ComponentSystemEvent;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    public List<Author> getAuthors() {
        ResultSet rs = null;
        PreparedStatement pst = null;
        Connection con = getConnection();
        String stm = "Select * from authors";
        List<Author> records = new ArrayList<Author>();
        try {
            pst = con.prepareStatement(stm);
            pst.execute();
            rs = pst.getResultSet();

            while (rs.next()) {
                Author author = new Author();
                author.setId(rs.getInt(1));
                author.setName(rs.getString(2));
                records.add(author);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            close(rs);
            close(pst);
            close(con);
        }
        return records;
    }
}
```
public Connection getConnection()
{
    Connection con = null;

    String url = "jdbc:postgresql://localhost/testdb";
    String user = "user1";
    String password = "user1";
    try {
        con = DriverManager.getConnection(url, user, password);
        System.out.println("Connection completed.");
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
    }
    finally{
    }
    return con;
}
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.
Spring provides special class DelegatingVariableResolver to integrate JSF and Spring together in a seamless manner.

Following steps are required to integrate Spring Dependency Injection (IOC) feature in JSF.

**Step 1: Add DelegatingVariableResolver**
Add a variable-resolver entry in faces-config.xml to point to spring class `DelegatingVariableResolver`.

```xml
<faces-config>
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
    ...
  </application>
</faces-config>
```

**Step 2: Add Context Listeners**
Add `ContextLoaderListener` and `RequestContextListener` listener provided by spring framework in web.xml.

```xml
<web-app>
  ...
  <!-- Add Support for Spring -->
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <listener>
    <listener-class>
      org.springframework.web.context.request.RequestContextListener
    </listener-class>
  </listener>
  ...
</web-app>
```
Step 3: Define Dependency
Define bean(s) in applicationContext.xml which will be used as dependency in managed bean.

```xml
<beans>
  <bean id="messageService"
       class="com.tutorialspoint.test.MessageServiceImpl">
    <property name="message" value="Hello World!" />
  </bean>
</beans>
```

Step 4: Add Dependency
DelegatingVariableResolver first delegates value lookups to the default resolver of the JSF and then to Spring's WebApplicationContext. This allows one to easily inject spring-based dependencies into one's JSF-managed beans.

We've injected messageService as spring-based dependency here.

```xml
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>userData</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.UserData</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>messageService</property-name>
      <value>#{messageService}</value>
    </managed-property>
  </managed-bean>
</faces-config>
```

Step 5: Use Dependency

```java
//jsf managed bean
public class UserData {
    //spring managed dependency
    private MessageService messageService;

    public void setMessageService(MessageService messageService) {
        this.messageService = messageService;
    }
}
```
```java
    }

    public String getGreetingMessage(){
        return messageService.getGreetingMessage();
    }
```

### Example Application

Let us create a test JSF application to test spring integration.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name <em>helloworld</em> under a package <em>com.tutorialspoint.test</em> as explained in the <em>JSF - First Application</em> chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify <em>pom.xml</em> as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Create <em>faces-config.xml</em> in <em>WEB-INF</em> folder as explained below.</td>
</tr>
<tr>
<td>4</td>
<td>Modify <em>web.xml</em> as explained below.</td>
</tr>
<tr>
<td>5</td>
<td>Create <em>applicationContext.xml</em> in <em>WEB-INF</em> folder as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Create <em>MessageService.java</em> under package <em>com.tutorialspoint.test</em> as explained below.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7</td>
<td>Create <code>MessageServiceImpl.java</code> under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>8</td>
<td>Create <code>UserData.java</code> under package <code>com.tutorialspoint.test</code> as explained below.</td>
</tr>
<tr>
<td>9</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>10</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>11</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>12</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**pom.xml**

```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.tutorialspoint.test</groupId>
  <artifactId>helloworld</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>helloworld Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
    </dependency>
  </dependencies>
</project>
```
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-api</artifactId>
  <version>2.1.7</version>
</dependency>
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.1.7</version>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>3.1.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>3.1.2.RELEASE</version>
</dependency>
</dependencies>
<build>
  <finalName>helloworld</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.1</version>
    </plugin>
  </plugins>
</build>
<configuration>
    <source>1.6</source>
    <target>1.6</target>
</configuration>

<plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.6</version>
    <executions>
        <execution>
            <id>copy-resources</id>
            <phase>validate</phase>
            <goals>
                <goal>copy-resources</goal>
            </goals>
            <configuration>
                <outputDirectory>${basedir}/target/helloworld/resources</outputDirectory>
                <resources>
                    <resource>
                        <directory>src/main/resources</directory>
                        <filtering>true</filtering>
                    </resource>
                </resources>
            </configuration>
        </execution>
    </executions>
</plugin>

</plugins>
</build>
</project>
faces-config.xml

```xml
<?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"
    version="2.0">
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
  <managed-bean>
    <managed-bean-name>userData</managed-bean-name>
    <managed-bean-class>com.tutorialspoint.test.UserData</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>messageService</property-name>
      <value>${messageService}</value>
    </managed-property>
  </managed-bean>
</faces-config>
```

web.xml

```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>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <!-- Add Support for Spring -->
</web-app>
```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>

<listener>
  <listener-class>
    org.springframework.web.context.request.RequestContextListener
  </listener-class>
</listener>

<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>

<web-app>

applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
  "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>

  <bean id="messageService"
    class="com.tutorialspoint.test.MessageServiceImpl">
    <property name="message" value="Hello World!” />
  </bean>

</beans>
```

MessageService.java

```java
package com.tutorialspoint.test;

public interface MessageService {
    String getGreetingMessage();
}
```

MessageServiceImpl.java
package com.tutorialspoint.test;

public class MessageServiceImpl implements MessageService {

    private String message;

    public String getGreetingMessage() {
        return message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}

UserData.java

package com.tutorialspoint.test;

import java.io.Serializable;

public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;

    private MessageService messageService;

    public MessageService getMessageService() {
        return messageService;
    }

    public void setMessageService(MessageService messageService) {
        this.messageService = messageService;
    }
}
public String getGreetingMessage(){
    return messageService.getGreetingMessage();
}

home.xhtml

<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
    </h:head>
    <h:body>
        <h2>Spring Integration Example</h2>
        #{userData.greetingMessage}
    </h:body>
</html>

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![Image of Spring Integration Example](image_url)

Spring Integration Example
Hello World!
JSF provides a rich expression language. We can write normal operations using \#{operation-expression} notation. Following are some of the advantages of JSF Expression languages.

- Can reference bean properties where bean can be an object stored in request, session or application scope or is a managed bean.
- Provides easy access to elements of a collection which can be a list, map or an array.
- Provides easy access to predefined objects such as a request.
- Arithmetic, logical and relational operations can be done using expression language.
- Automatic type conversion.
- Shows missing values as empty strings instead of NullPointerException.

**Example Application**
Let us create a test JSF application to test expression language.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Modify UserData.java under package com.tutorialspoint.test as explained below.</td>
</tr>
<tr>
<td>3</td>
<td>Modify home.xhtml as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>4</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
</tbody>
</table>
Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.

Launch your web application using appropriate URL as explained below in the last step.

**UserData.java**

```java
package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.Date;

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

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

private static final long serialVersionUID = 1L;

private Date createTime = new Date();
private String message = "Hello World!";

public Date getCreateTime() {
    return createTime;
}

public String getMessage() {
    return message;
}
}
```
home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>JSF Tutorial!</title>
    </h:head>
    <h:body>
        <h2>Expression Language Example</h2>
        Creation time:
        <h:outputText value="#{userData.createTime}"/>
        <br/>
        Message:
        <h:outputText value="#{userData.message}"/>
    </h:body>
</html>
```

Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![Expression Language Example](image)

Expression Language Example

Creation time: Sun Dec 09 22:29:35 IST 2012

Message: Hello World!
Internationalization is a technique in which status messages, GUI component labels, currency, date are not hardcoded in the program. Instead, they are stored outside the source code in resource bundles and retrieved dynamically. JSF provides a very convenient way to handle resource bundle.

Following steps are required to internalize a JSF application.

**Step 1: Define properties files**
Create properties file for each locale. Name should be in `<file-name>_<locale>.properties` format.

Default locale can be omitted in file name.

**messages.properties**

```
greeting=Hello World!
```

**messages_fr.properties**

```
greeting=Bonjour tout le monde!
```

**Step 2: Update faces-config.xml**

```
<application>
  <locale-config>
    <default-locale>en</default-locale>
    <supported-locale>fr</supported-locale>
  </locale-config>
  <resource-bundle>
    <base-name>com.tutorialspoint.messages</base-name>
    <var>msg</var>
  </resource-bundle>
</application>
```
Step 3: Use resource-bundle var

home.xhtml

```xml
<h:outputText value="#{msg['greeting']}" />
```

Example Application

Let us create a test JSF application to test internationalization in JSF.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a project with a name helloworld under a package com.tutorialspoint.test as explained in the JSF - First Application chapter.</td>
</tr>
<tr>
<td>2</td>
<td>Create resources folder under src -&gt; main folder.</td>
</tr>
<tr>
<td>3</td>
<td>Create com folder under src -&gt; main -&gt; resources folder.</td>
</tr>
<tr>
<td>4</td>
<td>Create tutorialspoint folder under src -&gt; main -&gt; resources -&gt; com folder.</td>
</tr>
<tr>
<td>5</td>
<td>Create messages.properties file under src -&gt; main -&gt; resources -&gt; com -&gt; tutorialspoint folder. Modify it as explained below.</td>
</tr>
<tr>
<td>6</td>
<td>Create messages_fr.properties file under src -&gt; main -&gt; resources -&gt; com -&gt; tutorialspoint folder. Modify it as explained below.</td>
</tr>
<tr>
<td>7</td>
<td>Create faces-config.xml in WEB-INF folder as explained below.</td>
</tr>
<tr>
<td>8</td>
<td>Create UserData.java under package com.tutorialspoint.test as explained below.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>9</td>
<td>Modify <code>home.xhtml</code> as explained below. Keep the rest of the files unchanged.</td>
</tr>
<tr>
<td>10</td>
<td>Compile and run the application to make sure the business logic is working as per the requirements.</td>
</tr>
<tr>
<td>11</td>
<td>Finally, build the application in the form of war file and deploy it in Apache Tomcat Webserver.</td>
</tr>
<tr>
<td>12</td>
<td>Launch your web application using appropriate URL as explained below in the last step.</td>
</tr>
</tbody>
</table>

**messages.properties**

```properties
messages.properties

greeting=Hello World!
```

**messages_fr.properties**

```properties
messages_fr.properties

greeting=Bonjour tout le monde!
```

**faces-config.xml**

```xml
<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">
   <application>
     <locale-config>
       <default-locale>en</default-locale>
       <supported-locale>fr</supported-locale>
     </locale-config>
     <resource-bundle>
```

336
<base-name>com.tutorialspoint.messages</base-name>
<var>msg</var>
</resource-bundle>
</application>
</faces-config>

UserData.java

package com.tutorialspoint.test;

import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Locale;
import java.util.Map;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {

    private static final long serialVersionUID = 1L;
    private String locale;
    private static Map<String, Object> countries;

    static {
        countries = new LinkedHashMap<String, Object>();
        countries.put("English", Locale.ENGLISH);
        countries.put("French", Locale.FRENCH);
    }

    public Map<String, Object> getCountries() {
        return countries;
    }
}
public String getLocale() {
    return locale;
}

public void setLocale(String locale) {
    this.locale = locale;
}

//value change event listener
public void localeChanged(ValueChangeEvent e){
    String newLocaleValue = e.getNewValue().toString();
    for (Map.Entry<String, Object> entry : countries.entrySet()) {
        if(entry.getValue().toString().equals(newLocaleValue)){
            FacesContext.getCurrentInstance()
                .getViewRoot().setLocale((Locale)entry.getValue());
        }
    }
}

home.xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>JSF tutorial</title>
  </h:head>
  <h:body>
    <h2>Internalization Language Example</h2>
    <h:form>
      <h:outputText value="#{msg['greeting']}" />
    </h:form>
  </h:body>
</html>
```
Once you are ready with all the changes done, let us compile and run the application as we did in JSF - First Application chapter. If everything is fine with your application, this will produce the following result.

![JSF application output](image1)

Change language from dropdown. You will see the following output.

![JSF language change](image2)