

JSF Tags

Overview

@author R.L. Martinez, Ph.D.

Java EE 7 provides a comprehensive list of JSF tags to support JSF web development. The tags are represented in XHTML format on the server and are converted into HTML and sent back to the client for rendering. These tags are included in the libraries that will be added to your projects during development.

This tutorial will cover a number of useful JSF tags. For a complete listing of available JSF tags consult the Oracle documentation at:

<http://docs.oracle.com/javaee/7/javaxserverfaces/2.2/vlddocs/facelets/>

<http://docs.oracle.com/javaee/6/javaxserverfaces/2.0/docs/pdldocs/facelets/index.html>

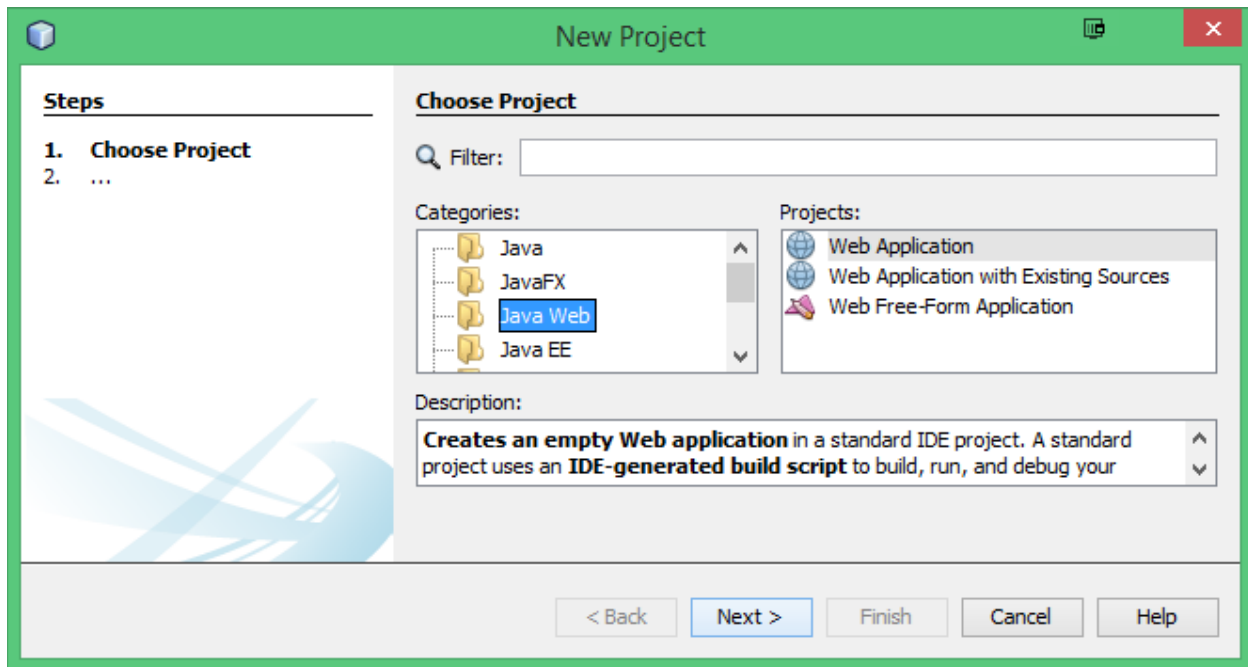
<http://docs.oracle.com/javaee/7/javaxserverfaces/2.0/docs/pdldocs/facelets/>

Let's start by building a project in NetBeans to which we will add and test a variety of JSF tags. Be prepared to carefully study this tutorial and devote the time necessary to understand the material. In web development, we work in multiple contexts (browser and servers) and therefore following code transitions can be challenging at times. Like any skill, your proficiency will improve with time and effort.

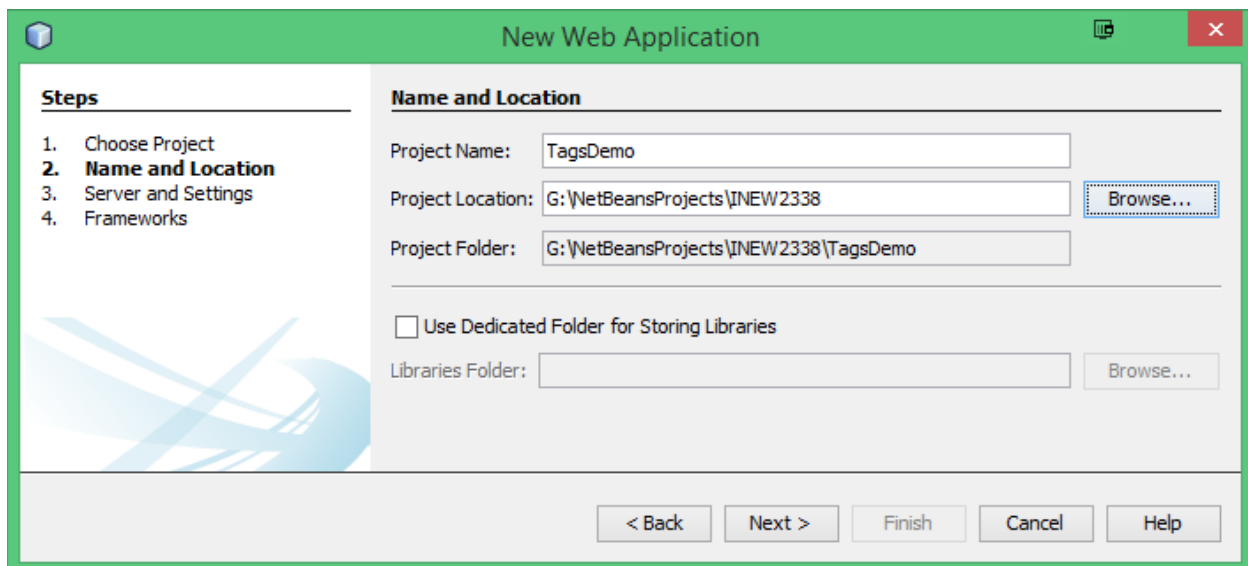
Starting with the TagsDemo Project

In NetBeans, select File | New Project and choose Java Web | Web Application and then Next.

JSF Tags

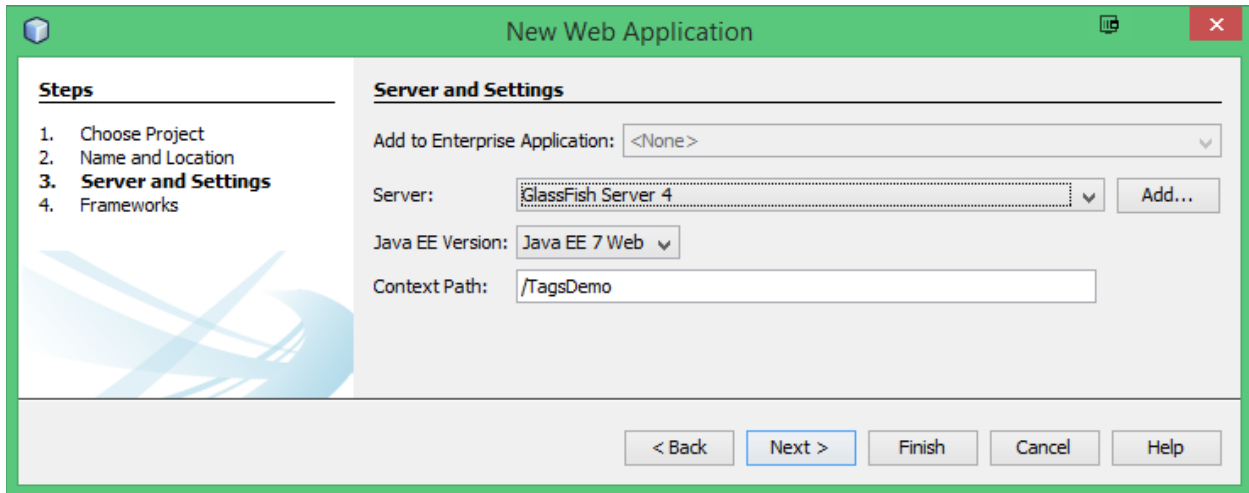


Name the project TagsDemo and modify the location and folder selection to reflect where you store your projects for the course. Select Next.



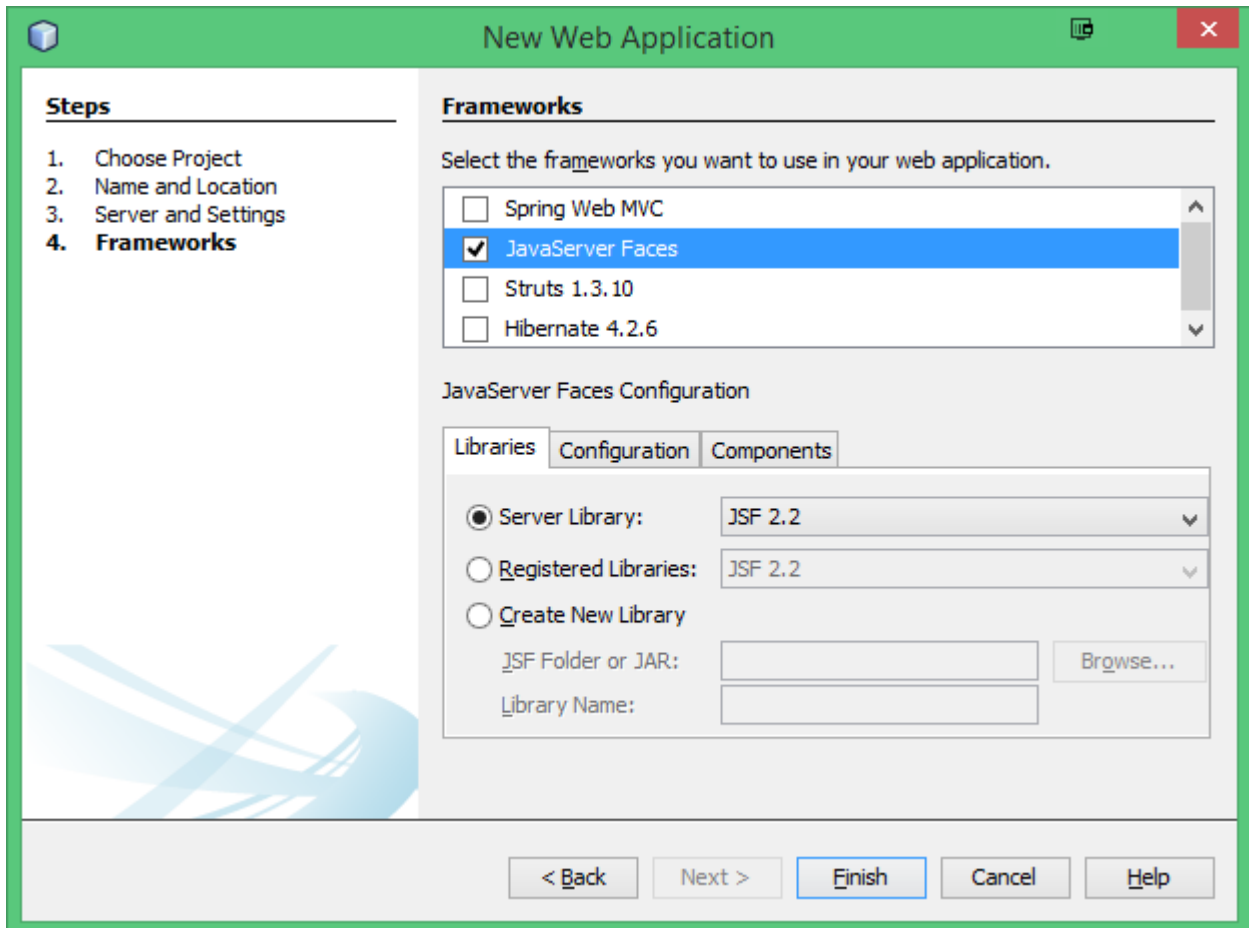
JSF Tags

No changes required below. Select Next.



The "New Web Application" dialog is shown with the "Server and Settings" tab selected. The "Steps" list on the left indicates the current step is "3. Server and Settings". The "Add to Enterprise Application" dropdown is set to "<None>". The "Server" dropdown is set to "GlassFish Server 4". The "Java EE Version" dropdown is set to "Java EE 7 Web". The "Context Path" text field contains "/TagsDemo". At the bottom, the "Next >" button is highlighted.

Check the JavaServer Pages box to add the JSF framework and then select Finish.



The "New Web Application" dialog is shown with the "Frameworks" tab selected. The "Steps" list on the left indicates the current step is "4. Frameworks". The "Select the frameworks you want to use in your web application." section shows a list of frameworks: "Spring Web MVC", "JavaServer Faces" (checked), "Struts 1.3.10", and "Hibernate 4.2.6". Below this, the "JavaServer Faces Configuration" section has three tabs: "Libraries", "Configuration", and "Components". The "Libraries" tab is active, showing three radio buttons: "Server Library:" (selected), "Registered Libraries:", and "Create New Library". The "Server Library:" dropdown is set to "JSF 2.2". The "Registered Libraries:" dropdown is also set to "JSF 2.2". The "Create New Library" section has two text fields: "JSF Folder or JAR:" and "Library Name:", each followed by a "Browse..." button. At the bottom, the "Finish" button is highlighted.

JSF Tags

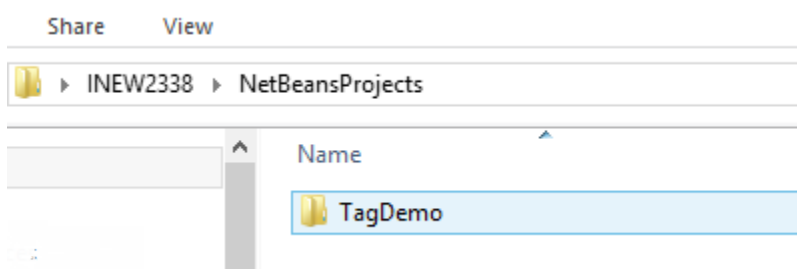
After selecting Finish, the project is built and the index.xhtml appears with the following minimal content.

```

1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:h="http://xmlns.jcp.org/jsf/html">
6      <h:head>
7          <title>Facelet Title</title>
8      </h:head>
9      <h:body>
10         Hello from Facelets
11     </h:body>
12 </html>

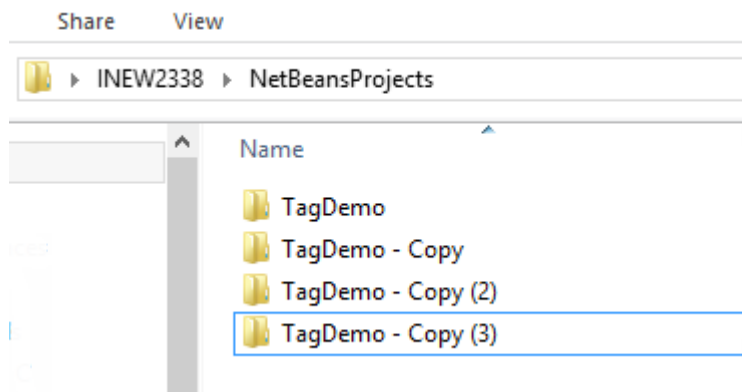
```

It is essential that software developers work to improve their organizational skills. Keeping files and folders organized and backed up is especially critical for web developers due to the additional complexity introduced by deploying and maintaining web-based solutions. Professional developers use software version repositories that help manage code projects. However, a simple technique like that described below can be used to backup projects and retain previous versions. Suppose I have the TagDemo project with which we are currently working in the folder G:\INEW2338\NetBeansProjects\ as shown.



A web developer not only wants to capture backups but we also want to retain previous versions of the code. We can accomplish both by the simple technique of R-clicking on the folder | Copy | Paste. The results of several copy and paste operations are shown. The developer continues to work with TagDemo.

JSF Tags



If at some point a developer would like to revert to a previous version of the project (usually the most recent), s/he would delete the corrupt TagDemo and rename TagDemo – Copy (3) to TagDemo and continue working with it. Better solutions such as the GIT source code management system exists but simple copy and paste operations will suffice for now.

Basic JSF Tags

The JSF basic tags provide coverage for many of the common web control functionality requirements such as: checkboxes, radio buttons, links, textareas, and quite a few more. Each of the JSF basic tags are processed by the servlet container and sent back to the client browser as standard HTML elements. The conversion results from JSF tag to HTML element can be viewed by R-clicking a page and selecting View Source after it has been rendered. Let's begin by adding and discussing a few example controls.

Modify index.xhtml as shown.

JSF Tags

```

1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!--index.xhtml-->
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5  <html xmlns="http://www.w3.org/1999/xhtml"
6  xmlns:f="http://java.sun.com/jsf/core"
7  xmlns:h="http://xmlns.jcp.org/jsf/html">
8  <h:head>
9  <title>JSF Tag Demo</title>
10 </h:head>
11 <h:body>
12 <!--JSF forms do not contain an action attribute since the JSF components
13 specify action attributes per the commandButton below or via other calls.-->
14 <h:form id="tagDemoForm">
15 <h3>inputText: readonly="true"</h3>
16 <h:inputText value="#{textData.readOnlyTextData}" readonly="true"/>
17 <h3>inputText</h3>
18 <h:inputText value="#{textData.mutableTextdata}"/>
19 <h3>inputSecret</h3>
20 <h:inputSecret value="#{textData.passworddata}" reisplay="true"/>
21 <h3>inputTextarea</h3>
22 <h:inputTextarea style="float:left; margin-right:20px; background-color:#ccc"
23 rows="10" cols="17" value="#{textData.textAreadata}"/>
24 <h3>selectManyCheckbox</h3>
25 <h:selectManyCheckbox value="#{checkBoxData.data}">
26 <f:selectItem itemValue="1" itemLabel="itemLabel 1" />
27 <f:selectItem itemValue="2" itemLabel="itemLabel 2" />
28 <f:selectItem itemValue="3" itemLabel="itemLabel 3" />
29 </h:selectManyCheckbox>
30 <h3>selectOneRadio</h3>
31 <h:selectOneRadio value="#{radioButtonData.data}">
32 <f:selectItem itemValue="1" itemLabel="itemLabel 1" />
33 <f:selectItem itemValue="2" itemLabel="itemLabel 2" />
34 <f:selectItem itemValue="3" itemLabel="itemLabel 3" />
35 </h:selectOneRadio><br />
36 <!--commandButton submitting the form to the action location-->
37 <h:commandButton value="commandButton1" action="displayOutput"/>
38 <!--links navigate to the outcome location-->
39 <h:link style="margin-left: 20px" value="link" outcome="displayOutput"/>
40 <br /><br />
41 <h:panelGrid id="panel" columns="2" border="1" cellpadding="10" cellspacing="1">
42 <f:facet name="header">
43 <h:outputText value="outputText"/>
44 </f:facet>
45 <h:outputLabel value="outputLabel - username"/>
46 <h:inputText value="#{textData.mutableTextdata}"/>
47 <h:outputLabel value="outputLabel - password" />
48 <h:inputSecret value="#{textData.passworddata}"/>
49 <f:facet name="footer">
50 <h:panelGroup style="display:block; text-align:center; background-color:orange;">
51 <h:commandButton value="commandButton2"
52 action="#{textData.setMutableTextdata('mutable')}" />
53 </h:panelGroup>
54 </f:facet>
55 </h:panelGrid>
56 <br />
57 <!--commandButton calling a method in a managed bean-->
58 <h:commandButton value="commandButton3"
59 action="#{textData.setTextAreadataAction('testing 123')}" />
60 </h:form>
61 </h:body>
62 </html>

```

JSF Tags

The references on lines 6 and 7 provide namespace coverage and recognition for the basic JSF tags in this example. Namespaces are primarily used to avoid name collisions. There are a number of JSF tags used in index.xhtml and most are discussed below. JSF tags are designed to render the HTML markup of its corresponding element. For instance, the h:head and h:body tags (lines 8-10 and 11-60) will render HTML <head> and <body> elements.

A number of user input controls are included on the page and within the form element. The value attributes of the UI controls are bound to data members of Java classes using Expression Language (EL). EL provides the communication link between UI components and their corresponding application logic. EL syntax is `{class.method or class.datamember}`. An example of EL can be seen on line 16: `{textData.readOnlyTextData}` where textData is the class (or object) and readOnlyTextData is a data member of that object.

The h:form tag on line 14 performs the role of the HTML <form> element. However, there is an important distinction between the JSF h:form tag and the HTML <form> element. HTML forms are processed after the submit operation (typically via a submit button). The form processing location (or file) is specified by the action attribute. Form data is submitted to the form processor via either the POST or GET method as specified by the method attribute.

On the other hand, JSF h:commandButton controls can not only serve as submit buttons as in line 37 but controls can also be bound to component data or functionality. For instance, each of the JSF input controls are bound to private data in the TextData.java class. When data changes in the object, data in bound controls is updated.

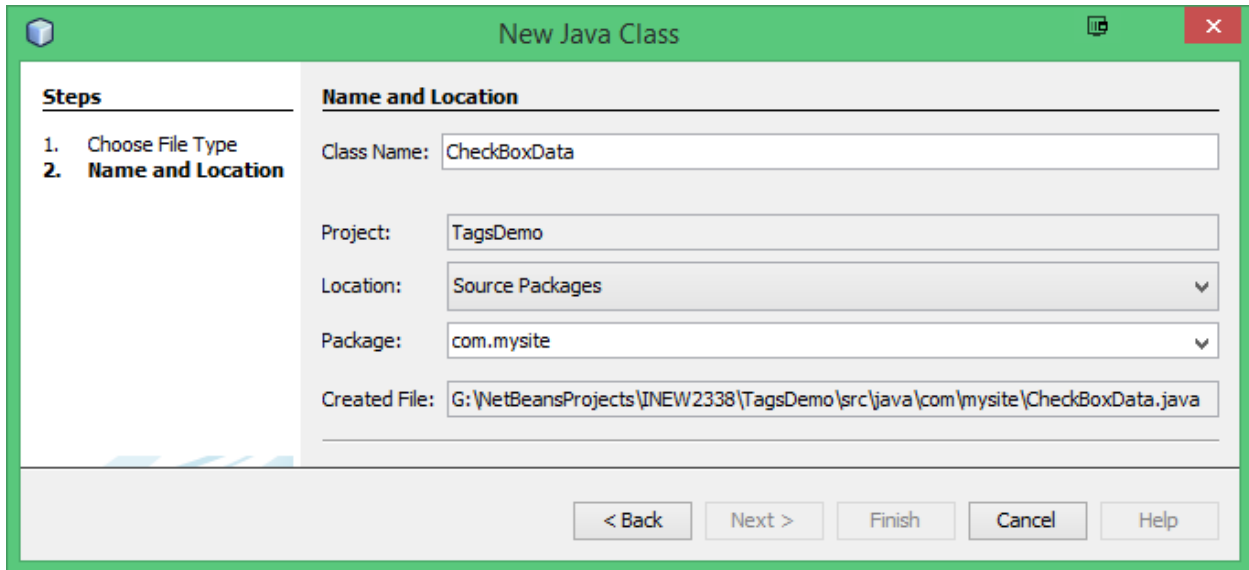
Also, notice the h:commandButton on lines 57-58. The action is set to call the setTextAreadataAction method of the textData.java component. When the button is clicked, the textAreadata is changed in the object by a call to the method setTextAreadataAction(). The data change is reflected in the h:inputTextarea control on lines 22-23. A call to setTextAreadata() would have performed the same update but the Action function was included to demonstrate an additional function beyond the mandatory getters and setters.

JSF Tags

Other controls on the page include the `h:selectManyCheckbox` and the `h:selectOneRadio`. Each of these is referencing their own managed bean (`checkBoxData.java` and `radioButtonData.java` respectively). Now we need to code the three managed beans used in the application.

By the way, in professional practice, I recommend that id attributes be set for all components. JSF will automatically generate cryptic ids for components without an id set. The component ids can be viewed by R-clicking a page and selecting View Source after it has been rendered. Id's are not included on all components in the course to help reduce code on the page.

Add a Java class by R-clicking TagsDemo | New | Java Class... and make the selections shown. Then click Finish.



Steps	Name and Location
1. Choose File Type	Class Name: <input type="text" value="CheckBoxData"/>
2. Name and Location	Project: <input type="text" value="TagsDemo"/>
	Location: <input type="text" value="Source Packages"/>
	Package: <input type="text" value="com.mysite"/>
	Created File: <input type="text" value="G:\NetBeansProjects\INEW2338\TagsDemo\src\java\com\mysite\CheckBoxData.java"/>

< Back Next > Finish Cancel Help

Code `CheckBoxData.java` as shown.

JSF Tags

```

1  ▢ /* CheckBoxData.java */
2    package com.mysite;
3
4  ▢ import java.io.Serializable;
5    | import javax.inject.Named;
6    | import javax.enterprise.context.RequestScoped;
7
8    @Named(value = "checkBoxData")
9    @RequestScoped
10
11   public class CheckBoxData implements Serializable {
12
13       private String[] data = {"1", "3"};
14
15   ▢   public String[] getData() {
16       |       return data;
17       |   }
18
19   ▢   public void setData(String[] data) {
20       |       this.data = data;
21       |   }
22   }

```

Repeat the previous steps to create the `RadioButtonData.java` and `TextData.java` classes in the `com.mysite` package.

Code `RadioButtonData.java` as shown.

JSF Tags

```
1  /* RadioButtonData.java */
2  package com.mysite;
3
4  import java.io.Serializable;
5  import javax.inject.Named;
6  import javax.enterprise.context.RequestScoped;
7
8  @Named(value = "radioButtonData")
9  @RequestScoped
10
11  public class RadioButtonData implements Serializable {
12
13      private String data = "2";
14
15      public String getData() {
16          return data;
17      }
18
19      public void setData(String data) {
20          this.data = data;
21      }
22  }
```

Code TextData.java as shown.

JSF Tags

```

1  □ /* TextData.java */
2  package com.mysite;
3
4  □ import java.io.Serializable;
5  | import javax.inject.Named;
6  | import javax.enterprise.context.RequestScoped;
7
8  @Named(value = "textData")
9  @RequestScoped
10
11 public class TextData implements Serializable {
12
13     private String readOnlyTextData = "This text is readonly.";
14     private String mutableTextdata = "This text can be changed.";
15     private String passworddata = "password";
16     private String textAreadata = "Everything is awesome!";
17
18     □ public String getReadOnlyTextData() {
19     |     return readOnlyTextData;
20     | }
21
22     □ public void setReadOnlyTextData(String readOnlyTextData) {
23     |     this.readOnlyTextData = readOnlyTextData;
24     | }
25
26     □ public String getMutableTextdata() {
27     |     return mutableTextdata;
28     | }
29
30     □ public void setMutableTextdata(String mutableTextdata) {
31     |     this.mutableTextdata = mutableTextdata;
32     | }
33
34     □ public String getPassworddata() {
35     |     return passworddata;
36     | }
37
38     □ public void setPassworddata(String passworddata) {
39     |     this.passworddata = passworddata;
40     | }
41
42     □ public String getTextAreadata() {
43     |     return textAreadata;
44     | }
45
46     ▼ □ public void setTextAreadata(String textAreadata) {
47     |     this.textAreadata = textAreadata;
48     | }
49
50     □ public void setTextAreadataAction(String textAreadata) {
51     |     this.textAreadata = textAreadata;
52     | }
53 }

```

JSF Tags

The last file for this project is displayOutput.xhtml. Add it to the project by R-clicking TagsDemo | New | XHTML. Enter code for the file as shown.

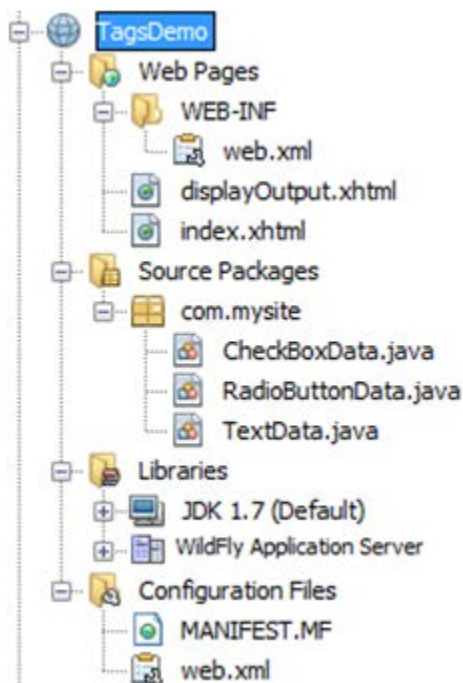
```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--displayOutput.xhtml-->
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5  <html xmlns="http://www.w3.org/1999/xhtml"
6  xmlns:h="http://java.sun.com/jsf/html"
7  xmlns:ui="http://java.sun.com/jsf/facelets">
8  <h:body>
9      <h2>Display Text Data</h2>
10     <hr />
11     #{textData.readOnlyTextData}<br />
12     #{textData.mutableTextdata}<br />
13     #{textData.passworddata}<br />
14     #{textData.textAreadata}<br />
15     <h2>Display Check Box Data</h2>
16     <hr />
17     <ui:repeat value="#{checkBoxData.data}" var="item">
18         #{item}
19     </ui:repeat>
20     <h2>Display Radio Button Data</h2>
21     <hr />
22     #{radioButtonData.data}
23     <hr />
24     <!--links navigate to the outcome location-->
25     <h:link value="index" outcome="index" />
26 </h:body>
27 </html>

```

The TagsDemo project should look like the following in the Projects window.

JSF Tags



Each of the classes added is a managed JavaBean (a.k.a managed bean). They supply server-side, object-based support for the components on the browser. The managed beans all have getter and setter methods which provide access to the private data members. Managed beans must also be serializable which enables persistent storage by the container if necessary. Lines 16-23 of index.xhtml demonstrate access to data members of the TextData class. The JSF input tags are linked directly (bound) to data members in the object. Other bound controls on the page include the h:selectManyCheckbox, h:selectOneRadio, and h:commandButton.

Note: The term bound is used here to describe the link between the HTML element (as rendered by the JSF tag) and the private data member of the referenced object. There is also a “binding” attribute of some JSF basic tags that can be set which binds to an entire object but that is not used in this tutorial.

Like the input tags, each of the other bound controls is linked to data in its associated object. When the page loads, the getter method of the object is automatically called for bound controls. On line 25, the h:selectManyCheckbox control is tied to the string array named data on line 13 in CheckBoxData.java. An

JSF Tags

array is required for this control since multiple selections can be made by the user. Note that the values 1 and 3 are used to select the first and third checkboxes when the page loads. The same behavior can be observed in `RadioButtonData.java` on line 13 where the value 2 is used to select the second radio button.

Other JSF tags on the `index.xhtml` page include the `h:panelGrid`, `f:facet`, and `h:panelGroup` which provide page layout options. Another important JSF tag which is not included in this tutorial is the `h:dataTable`. It is covered in the JDBC tutorial.

JSF Tags

R-click on index.xhtml in the Project window and select “Run File” to run the file. The output is shown. Note that the controls are populated with the default values of the private data members of the objects to which they are bound.

inputText: readonly="true"

This text is readonly.

inputText

This text can be changed.

inputSecret

.....

inputTextarea

Everything is awesome!

selectManyCheckbox

☒ itemLabel 1 ☐ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1

[link](#)

outputText	
outputLabel - username	This text can be changed.
outputLabel - password
<div>commandButton2</div>	

commandButton3

Next, select commandButton1 to see the output below.

JSF Tags

Display Text Data

This text is readonly.

This text can be changed.

Everything is awesome!

Display Check Box Data

1 3

Display Radio Button Data

2

[index](#)

Select the Back button to return to index.xhtml. Next, make a few changes to the values of the controls on index.xhtml and then click `commandButton1`. Notice that the changes are reflected in `displayOutput.xhtml` which is the action attribute value of `commandButton1` (see line 37 of index.xhtml).

JSF Tags

Display Text Data

This text is readonly.
I changed this.
Now the password appears!
And I changed that.

Display Check Box Data

1 2 3

Display Radio Button Data

1

[index](#)

In traditional web development, when another page is supplied as the value to the action attribute of a form, a submit action is performed. By submitting the form, all control name:value pairs are sent to a form processor destination as specified by the action attribute. For instance, this form processor action attribute could be JavaScript, PHP, or ASP.NET.

Notice the h:form control on line 14 of index.xhtml does not have an action attribute. Form submissions are performed in JSF via the h:commandButton (and h:commandLink) controls which do have action attributes. The displayOutput.xhtml page (a form processor) is called from line 37.

When an h:commandButton is clicked a POST request is generated and all setters for the data bindings on the form are called and the value of the action attribute is performed. In the case of commandButton1, the action attribute is set to navigate to displayOutput.xhtml. Alternatively, commandButton2 and 3 call methods in the TextData.java managed bean.

The action for commandButton2 calls the setMutableTextdata() method with an argument of “mutable” which sets the value of the h:inputText control. The action

JSF Tags

of `commandButton3` calls the `setTextAreaDataAction` method with an argument of “testing 123” which sets the value of the `h:inputTextarea` control.

A couple of points are very important and deserve restating. Both of these points are essential to understanding the way JSF controls operate. First, clicking an `h:commandButton` on a form calls the setters of all data members located on that form. If you wish to limit this behavior to specific controls, then use multiple forms. Second, clicking an `h:commandButton` not only performs the action if it is a method call as in the case of `commandButton2` and `3`, it also performs a POST of the form which calls the setters of all data members on the form.

To observe other important behavior, navigate back to `index.xhtml` and click “link” which is to the right of `commandButton1`. Selecting link issues a GET request for the value of the outcome attribute which is `displayOutput` in this case. By clicking “link” we confirm that the **initial** values of data members are displayed. On the other hand, when `commandButton1` is clicked, the data changes made on `index.xhtml` are reflected in `displayOutput.xhtml` since a POST is performed and the data setters are called.

The initial values are displayed with the link control for two reasons: a GET request was issued and all managed beans in this tutorial have their scope set to `@RequestScoped`. Line 9 (`@RequestScoped`) specifies that the lifetime of the bean is limited to a single request. If we wanted data values to endure longer (e.g. while the session is active), we could use `@SessionScoped`. More scope alternatives are reviewed in an upcoming module.

Steps to Enhance JSF Control Understanding

Before beginning the practice steps, it may be helpful to increase the session timeout duration of your application. The default is 30 minutes which will likely produce a view timeout error like the following during development and testing.

An Error Occurred:

viewId:/index.xhtml - View /index.xhtml could not be restored.

JSF Tags

To minimize the view timeout errors, increase the session-timeout value in web.xml to a value higher than 30 minutes. It is increased to 330 in the excerpt below.

```
<session-config>
  <session-timeout>
    330
  </session-timeout>
</session-config>
```

Perform these steps to observe important form, variable, control, and method behavior. You will likely benefit by working through the examples several times since the concepts are a bit involved. It is not necessary to develop perfect understanding of the steps and responses at this point. Rather, it is important to diligently consider the code and event responses. Also, you are encouraged to code the examples and observe the behavior directly.

1. R-click index.xhtml in the Project window to run the file which produces output as shown. Verify that all controls are set to initial values obtained by executing the getters for each EL data reference. See the class files to confirm the initial values.

JSF Tags

inputText: readonly="true"

This text is readonly.

inputText

This text can be changed.

inputSecret

.....

inputTextarea

Everything is
awesome!

selectManyCheckbox

☒ itemLabel 1 ☐ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

outputText	
outputLabel - username	<input type="text" value="This text can be changed."/>
outputLabel - password	<input type="password"/>
<div>commandButton2</div>	

commandButton3

- Make the following changes to the form. Change the content of the second text box to “HELLO”, the text in the text area to “SUPER!”, and select all three check boxes. Make the changes as shown.

JSF Tags

inputText: readonly="true"

This text is readonly.

inputText

HELLO

inputSecret

.....

inputTextarea

SUPER!

selectManyCheckbox

☒ itemLabel 1 ☒ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

outputText	
outputLabel - username	<input type="text" value="This text can be changed."/>
outputLabel - password	<input type="password"/>
<div>commandButton2</div>	

commandButton3

JSF Tags

- After making the changes above, click `commandButton1` to submit the form to `displayOutput.xhtml` which is shown. Notice that **SUPER!** and the second check box were submitted but the text in the text box was not changed to **HELLO**. Do you know why? Here's a hint: It is the same reason that the password is not displayed. Pause a few minutes and see if you can answer that... Well, the text for that text box actually was changed.

However, the `h:inputText` controls on line 18 and on line 46 of `index.xhtml` are bound to the same value. The default value still contained in line 46 was used to populate the EL on line 12 of `displayOutput.xhtml`. Observe the other EL contained in `displayOutput.xhtml` and ensure that you understand the corresponding relationship between the EL and the output shown.

Display Text Data

This text is readonly.

This text can be changed.

SUPER!

Display Check Box Data

1 ☒ 3

Display Radio Button Data

2

[index](#)

- Select the Back button on the page to navigate back to `index.xhtml` which is shown. Notice that the changes are retained since most browsers use cache (file or in-memory) content for the Back button.

JSF Tags

inputText: readonly="true"

This text is readonly.

inputText

HELLO

inputSecret

.....

inputTextarea

SUPER!

selectManyCheckbox

☒ itemLabel 1 ☒ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

outputText	
outputLabel - username	<input type="text" value="This text can be changed."/>
outputLabel - password	<input type="password"/>
<div>commandButton2</div>	

commandButton3

- Now select “link” (to the right of commandButton1) which navigates to displayOutput.xhtml. Notice that all **initial** data from the classes is displayed.

JSF Tags

The changes previously made are not reflected. Why? Again, see if you can answer the question by inspecting the code... The reason the changes are not reflected is that the page was accessed via an `h:link` and not an `h:commandButton`. Also, notice that the value “password” is displayed but does not appear when `commandButton1` is pressed. Why? The same output is shown when `index.xhtml` is refreshed (confirm by selecting F5 in Chrome – more on this with item 7 below).

The `h:commandButton` controls submit a POST while an `h:link` simply navigates to the outcome. Furthermore, each of the classes have `@RequestScoped` specified on line 9 which means the objects only have request lifetime and do not endure across requests. If you wanted longer-lived data, you could use `@SessionScoped`. However, `@RequestScoped` is usually the best choice for UI bound data members.

Display Text Data

This text is readonly.
 This text can be changed.
 password
 Everything is awesome!

Display Check Box Data

1 3

Display Radio Button Data

2

[index](#)

JSF Tags

6. Select the Back button to see the original page with changes again. Recall that the Back button does not force a request to the server but rather is rendered from cache.

inputText: readonly="true"

This text is readonly.

inputText

HELLO

inputSecret

.....

inputTextarea

SUPER!

selectManyCheckbox

☒ itemLabel 1 ☒ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

outputText	
outputLabel - username	This text can be changed.
outputLabel - password	
<div>commandButton2</div>	

commandButton3

JSF Tags

7. Select F5 to refresh the page and see the beginning page again. When F5 is clicked in Chrome (the browser used in this tutorial) it issues a “Cache-Control: max-age=0” header in the request to the server which forces a reload of the page. Notice the controls are set to the initial values specified in the classes. The same header does not force a reload in Firefox.

To obtain the full “refresh” behavior in Firefox, the Ctrl-F5 combination is required which adds headers “Cache-Control: no-cache” and “Pragma: no-cache” to the request. Browser variations introduce complications and can make web development particularly challenging. Fortunately in this course, accounting for browser deviations is not required.

JSF Tags

inputText: readonly="true"

inputText

inputSecret

inputTextarea

Everything is awesome!

selectManyCheckbox

☒ itemLabel 1
 ☐ itemLabel 2
 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1
 ☒ itemLabel 2
 ☐ itemLabel 3

[link](#)

outputText	
outputLabel - username	<input type="text" value="This text can be changed."/>
outputLabel - password	<input type="password"/>
<div style="background-color: orange; display: inline-block; padding: 5px 20px; border: 1px solid #ccc;"> commandButton2 </div>	

JSF Tags

8. Click `commandButton2` to see the output below. Notice that the text box value has been changed to “mutable” by the method call on line 52 which also changes the value of the control on line 18. Also notice that the first `h:inputSecret` control is now blank. Why? Based on previous examples, you have enough information to answer.

`inputText: readonly="true"`

This text is readonly.

`inputText`

mutable

`inputSecret`

`inputTextarea`

Everything is awesome!

`selectManyCheckbox`

☒ itemLabel 1 ☐ itemLabel 2 ☒ itemLabel 3

`selectOneRadio`

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

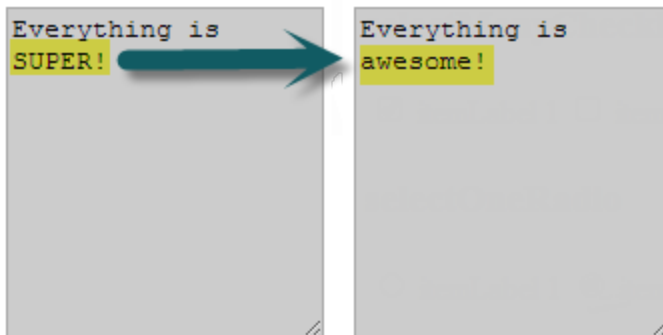
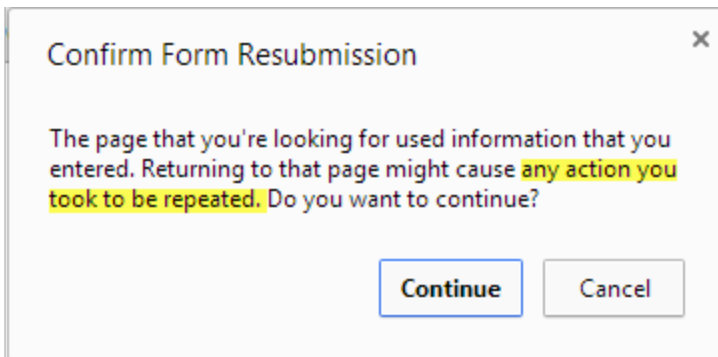
outputText	
outputLabel - username	mutable
outputLabel - password	
<div>commandButton2</div>	

commandButton3

JSF Tags

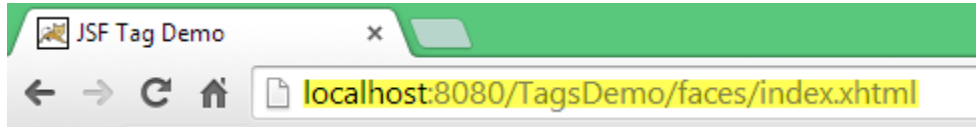
- Now replace the word awesome in the text area with SUPER and press F5 to refresh the page. SUPER becomes awesome again. Why? Selecting F5 produces the dialog shown. However, this dialog did not appear when F5 was pressed in step 7. Why? F5 in step 7 issued a GET request and pressing F5 “after” selecting `commandButton2` reissues the POST request that was sent by `commandButton2`.

A GET request does not produce the dialog box. See the highlight “any action you took to be repeated” in the dialog. The POST action and the call to method `setMutableTextdata` are both repeated. Recall from above that `h:commandButtons` submit POSTs in addition to another action specified by the action attribute (see line 52).



JSF Tags

10. Let's perform another test. Click in the address bar and press Enter which will force a full refresh (an original GET request to the server in this case).



11. Press `commandButton2` followed by `commandButton3` which will produce the output shown.

JSF Tags

inputText: readonly="true"

This text is readonly.

inputText

mutable

inputSecret

inputTextarea

testing 123


selectManyCheckbox

☒ itemLabel 1 ☐ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1 [link](#)

outputText	
outputLabel - username	mutable 
outputLabel - password	
<div>commandButton2</div>	

commandButton3

- Now change “testing 123” and the “mutable” text with the blue star to “Java EE is cool!” After making those two changes, press commandButton3

JSF Tags

to see the output below. Explain why both h:inputText controls retain the “Java EE is cool!” setting while the h:inputTextarea changes back to “testing 123”.

inputText: readonly="true"

This text is readonly.

inputText

Java EE is cool!

inputSecret

inputTextarea

testing 123

selectManyCheckbox

☒ itemLabel 1 ☐ itemLabel 2 ☒ itemLabel 3

selectOneRadio

☐ itemLabel 1 ☒ itemLabel 2 ☐ itemLabel 3

commandButton1

[link](#)

outputText	
outputLabel - username	Java EE is cool!
outputLabel - password	
<div>commandButton2</div>	

commandButton3

JSF Tags

13. Work through these steps and those of your own to ensure you understand JSF control behavior.

Facelets Tags

Facelets is a view definition language specification that supports creation of templates in JSF. Unlike other web development platforms, the JSF templates created with Facelets tags are dynamically constructed at runtime. The contents are inserted into templates instead of the design-time approach which uses the template to build all dependent pages during development. The Facelets runtime method dramatically reduces the HTML markup required by the site since the template is not replicated across all pages that use it.

Common Facelets tags are listed in the table. Most are demonstrated in a subsequent module.

Tag	Purpose
ui:insert	Specifies template editable region (in template)
ui:define	Defines the content to be inserted (in client)
ui:composition	Specifies template to use or default content
ui:include	Specifies file to include
ui:param	Used to pass named objects between Facelets

Converter Tags

Converter tags provide methods to perform common conversions of strings to dates and strings to numbers. The date and number converter tags are demonstrated in a subsequent module.

Tag	Purpose
f:convertDateTime	Converts String to Date of specified format
f:convertNumber	Converts String to Number of specified format
Custom	Used to create a custom converter

JSF Tags

Validation Tags

Validation tags provide system-level checks for common validation tasks. See the links at the top of this tutorial for examples of validation tag implementations.

Tag	Purpose
f:validateLength	Validates String length
f:validateLongRange	Validates numeric range of a long value
f:validateDoubleRange	Validates numeric range of a float value
f:validateRegex	Validates regular expressions
Custom	Used to create a custom validator