Introduction to Facelets

The term Facelets refers to the view declaration language for JavaServer Faces technology.

JavaServer Pages (JSP) technology, previously used as the presentation technology for JavaServer Faces, does not support all the new features available in JavaServer Faces 2.0.

JSP technology is considered to be a deprecated presentation technology for JavaServer Faces 2.0.

Facelets is a part of the JavaServer Faces specification and also the preferred presentation technology for building JavaServer Faces technology-based applications.

The following topics are addressed here:

- What Is Facelets?
- Developing a Simple Facelets Application
- Templating
- Composite Components
- Resources

What Is Facelets?

Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces views using HTML style templates and to build component trees.

Facelets features include the following:

- Use of XHTML for creating web pages
- Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries
- Support for the Expression Language (EL)
- Templating for components and pages

Advantages of Facelets for large-scale development projects include the following:

- Support for code reuse through templating and composite components
- Functional extensibility of components and other server-side objects through customization
- Faster compilation time
- Compile-time EL validation
- High-performance rendering

In short, the use of Facelets reduces the time and effort that needs to be spent on development and deployment.

Facelets views are usually created as XHTML pages.

JavaServer Faces implementations support XHTML pages created in conformance with the XHTML Transitional Document Type Definition (DTD), as listed at http://www.w3.org/TR/xhtml1/#a_dtd_XHTML-1.0-Transitional.

By convention, web pages built with XHTML have an `.xhtml` extension.

JavaServer Faces technology supports various tag libraries to add components to a web page.

To support the JavaServer Faces tag library mechanism, Facelets uses XML namespace declarations.

Table 5-1 lists the tag libraries supported by Facelets.

Table 5-1 Tag Libraries Supported by Facelets

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	`http://java.sun.com/jsf/facelets`	`ui:`	`ui:component` `ui:insert`	Tags for templating

JavaServer Faces HTML Tag Library	`http://java.sun.com/jsf/html`	`h:`	`h:head`
			`h:body`
			`h:outputText`
			`h:inputText`
				JavaServer Faces component tags for all `UIComponents`

JavaServer Faces Core Tag Library	`http://java.sun.com/jsf/core`	`f:`	`f:actionListener` `f:attribute`	Tags for JavaServer Faces custom actions that are independent of any particular `RenderKit`
JSTL Core Tag Library	`http://java.sun.com/jsp/jstl/core`	`c:`	`c:forEach` `c:catch`	JSTL 1.1 Core Tags
JSTL Functions Tag Library	`http://java.sun.com/jsp/jstl/functions`	`fn:`	`fn:toUpperCase` `fn:toLowerCase`	JSTL 1.1 Functions Tags

In addition, Facelets supports tags for composite components for which you can declare custom prefixes.

For more information on composite components, see Composite Components.

Based on the JavaServer Faces support for Expression Language (EL) syntax defined by JSP 2.1, Facelets uses EL expressions to reference properties and methods of backing beans.

EL expressions can be used to bind component objects or values to methods or properties of managed beans.

For more information on using EL expressions, see Using the EL to Reference Backing Beans.

Developing a Simple Facelets Application

This section describes the general steps involved in developing a JavaServer Faces application.

The following tasks are usually required:

- Developing the backing beans
- Creating the pages using the component tags
- Defining page navigation
- Mapping the `FacesServlet` instance
- Adding managed bean declarations

Creating a Facelets Application

The example used in this tutorial is the `guessnumber` application.

The application presents you with a page that asks you to guess a number between 0 and 10, validates your input against a random number, and responds with another page that informs you whether you guessed the number correctly or incorrectly.

Developing a Backing Bean

In a typical JavaServer Faces application, each page of the application connects to a backing bean, a type of managed bean.

The backing bean defines the methods and properties that are associated with the components.

The following managed bean class, UserNumberBean.java, generates a random number from 0 to 10:

```
package guessNumber;
import java.util.Random;
import
javax.faces.bean.ManagedBean;
import
javax.faces.bean.SessionScoped;
```

```
@ManagedBean
@SessionScoped
public class UserNumberBean {
Integer randomInt = null;
Integer userNumber = null;
String response = null;
private long maximum=10;
private long minimum=0;
public UserNumberBean() {
Random randomGR = new Random();
randomInt =
new Integer(randomGR.nextInt(10));
System.out.println
("Duke's number: " + randomInt);
}
public void setUserNumber
(Integer user_number)
{ userNumber = user_number; }
public Integer getUserNumber()
{ return userNumber; }
```

```
public String getResponse() {
if ((userNumber != null) &&
(userNumber.compareTo(randomInt)
== 0))
{ return "Yay! You got it!"; }
else {
return "Sorry, " + userNumber +
" is incorrect.";
}
}
```

```
public long getMaximum()
{ return (this.maximum); }
public void setMaximum
(long maximum)
{ this.maximum = maximum; }
public long getMinimum()
{ return (this.minimum); }
public void setMinimum
(long minimum)
{ this.minimum = minimum; }
}
```

Note the use of the `@ManagedBean` annotation, which registers the backing bean as a resource with JavaServer Faces implementation.

The `@SessionScoped` annotation registers the bean scope as `session`.

Creating Facelets Views

Creating a page or view is the responsibility of a page author.

This task involves adding components on the pages, wiring the components to backing bean values and properties, and registering converters, validators, or listeners onto the components.

For the example application, XHTML web pages serve as the front end.

The first page of the example application is a page called `greeting.xhtml`.

A closer look at various sections of this web page provides more information.

The first section of the web page declares the content type for the page, which is XHTML:

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
```

The next section declares the XML namespace for the tag libraries that are used in the web page:

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

The next section uses various tags to insert components into the web page:

```
<h:head>
<title>
Guess Number Facelets Application
</title>
</h:head>
<h:body>
<h:form>
```

```
<h:graphicImage
value="#{resource['images:wave.med.
gif']}"/>
<h2>
Hi, my name is Duke.

I am thinking of a number from
#{userNumberBean.minimum} to
#{userNumberBean.maximum}.
Can you guess it?
<p></p>
```

```
<h:inputText
id="userNo"
value=
"#{userNumberBean.userNumber}"
>
<f:validateLongRange
minimum="#{userNumberBean.minimum}"
maximum="#{userNumberBean.maximum}"
/>
</h:inputText>
```

```
<h:commandButton
id="submit" value="Submit"
action="response.xhtml"/>
<h:message showSummary="true"
showDetail="false"
style="
color: red;
font-family:
'New Century Schoolbook', serif;
font-style: oblique;
text-decoration: overline"
```

```
id="errors1"
for="userNo"/>
</h2>
</h:form>
</h:body>
```

Note the use of the following tags:

- Facelets HTML tags (those beginning with `h:`) to add components
- The Facelets core tag `f:validateLongRange` to validate the user input

An `inputText` component accepts user input and sets the value of the backing bean property `userNumber` through the EL expression `#{userNumberBean.userNumber}`.

The input value is validated for value range by the JavaServer Faces standard validator `f:validateLongRange`.

The image file, `wave.med.gif`, is added to the page as a resource.

For more details about the resources facility, see Resources.

A `commandButton` component with the ID `submit` starts validation of the input data when a user clicks the button.

Using implicit navigation, the component redirects the client to another page, `response.xhtml`, which shows the response to your input.

You can now create the second page, `response.xhtml`, with the following content:

```
<!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:head>
<title>
Guess Number Facelets Application
</title>
</h:head>
<h:body>
<h:form>
<h:graphicImage
value=
"#{resource['images:wave.med.gif']}"
/>
```

```
<h2>
<h:outputText id="result"
value="#{userNumberBean.response}"
/>
</h2>
<h:commandButton id="back"
value="Back"
action="greeting.xhtml"/>
</h:form>
</h:body>
</html>
```

Configuring the Application

Configuring a JavaServer Faces application involves mapping the Faces Servlet in the web deployment descriptor file, such as a `web.xml` file, and possibly adding managed bean declarations, navigation rules, and resource bundle declarations to the application configuration resource file, `faces-config.xml`.

If you are using NetBeans IDE, a web deployment descriptor file is automatically created for you.

In such an IDE-created `web.xml` file, change the default greeting page, which is `index.xhtml`, to `greeting.xhtml`.

Here is an example `web.xml` file, showing this change in bold.

```
<?xml version="1.0"
encoding="UTF-8"?>
<web-app version="3.0"
xmlns=
"http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/w
eb-app_3_0.xsd">
```

```
<context-param>
<param-name>
javax.faces.PROJECT_STAGE
</param-name>
<param-value>
Development
</param-value>
</context-param>
```

```
<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>
<session-config>
<session-timeout>
30
</session-timeout>
</session-config>
```

```
<welcome-file-list>
<welcome-file>
faces/greeting.xhtml
</welcome-file>
</welcome-file-list>
</web-app>
```

Note the use of the context parameter
PROJECT_STAGE.

This parameter identifies the status of a JavaServer Faces application in the software lifecycle.

The stage of an application can affect the behavior of the application.

For example, if the project stage is defined as `Development`, debugging information is automatically generated for the user.

If not defined by the user, the default project stage is `Production`.

Building, Packaging, Deploying, and Running the `guessnumber` Facelets Example

You can use either NetBeans IDE or Ant to build, package, deploy, and run the `guessnumber` example.

The source code for this example is available in the *tut-install*`/examples/web/guessnumber` directory.

To Build, Package, and Deploy the `guessnumber` Example Using NetBeans IDE

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:
   `tut-install/examples/web/`

3. Select the `guessnumber` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

6. In the Projects tab, right-click the `guessnumber` project and select Deploy.

This option builds and deploys the example application to your GlassFish Server instance.

To Build, Package, and Deploy the `guessnumber` Example Using Ant

1. In a terminal window, go to:

   `tut-install``/examples/web/guessnumber/`

2. Type the following command:

```
ant
```

This command calls the `default` target, which builds and packages the application into a WAR file, `guessnumber.war`, that is located in the `dist` directory.

3. Make sure that the GlassFish Server is started.

4. To deploy the application, type the following command: `ant deploy`

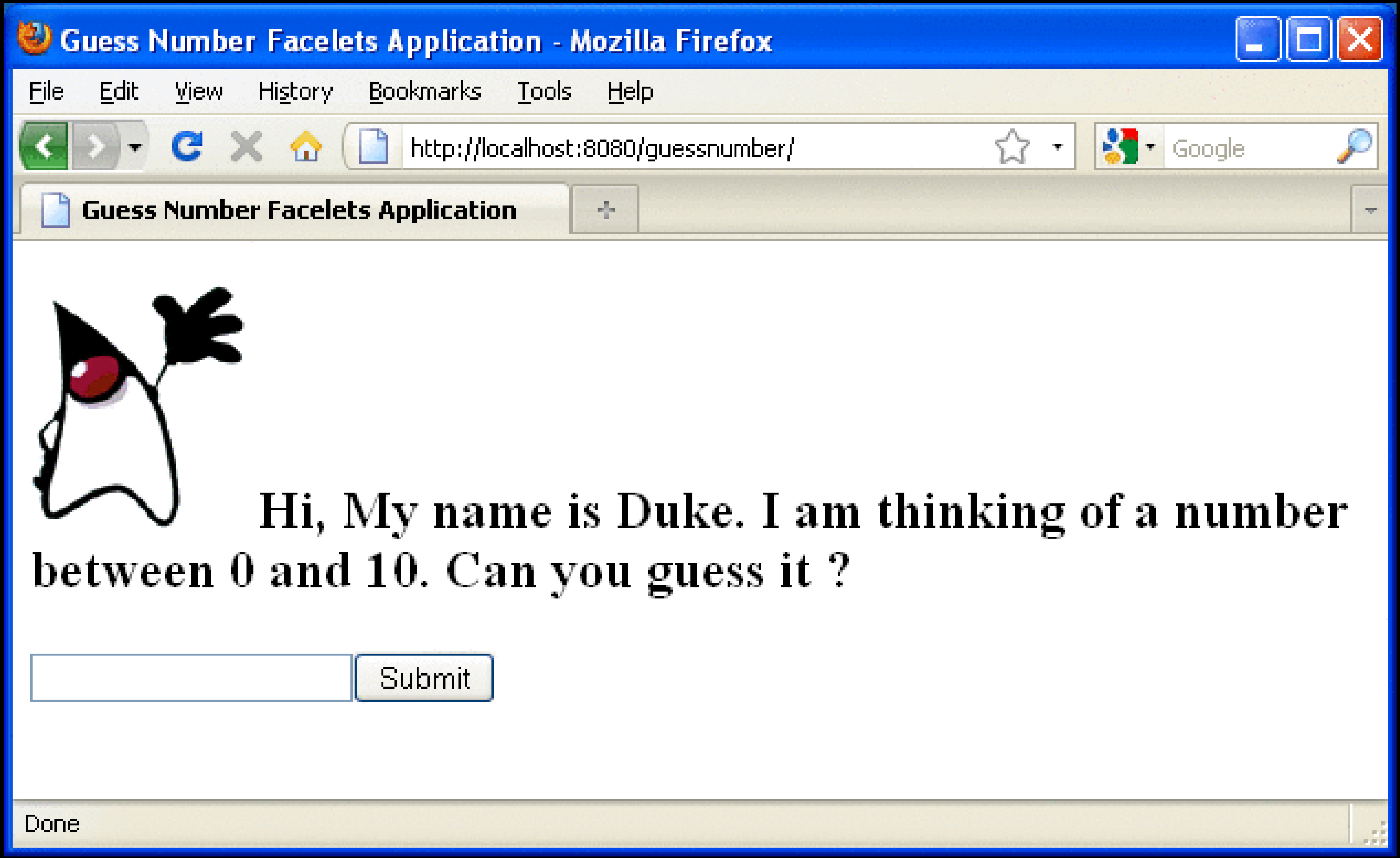
To Run the `guessnumber` Example

1. Open a web browser.

2. Type the following URL in your web browser:
   `http://localhost:8080/guessnumber`

The web page shown in Figure 5-1 appears.

Figure 5-1 Running the `guessnumber` Application


Guess Number Facelets Application - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/guessnumber/
Google
Guess Number Facelets Application
Submit
Done

Hi, My name is Duke. I am thinking of a number between 0 and 10. Can you guess it ?

3. In the text field, type a number from 0 to 10 and click Submit.

Another page appears, reporting whether your guess is correct or incorrect.

4. If you guessed incorrectly, click the Back button to return to the main page.

You can continue to guess until you get the correct answer.

Templating

JavaServer Faces technology provides the tools to implement user interfaces that are easy to extend and reuse.

Templating is a useful Facelets feature that allows you to create a page that will act as the base, or template, for the other pages in an application.

By using templates, you can reuse code and avoid recreating similarly constructed pages.

Templating also helps in maintaining a standard look and feel in an application with a large number of pages.

Table 5-2 lists Facelets tags that are used for templating and their respective functionality.

Table 5-2 Facelets Templating Tags

Tag	Function
`ui:component`	Defines a component that is created and added to the component tree.
`ui:composition`	Defines a page composition that optionally uses a template. Content outside of this tag is ignored.
`ui:debug`	Defines a debug component that is created and added to the component tree.
`ui:decorate`	Similar to the composition tag but does not disregard content outside this tag.
`ui:define`	Defines content that is inserted into a page by a template.

`ui:fragment`	Similar to the component tag but does not disregard content outside this tag.
`ui:include`	Encapsulate and reuse content for multiple pages.
`ui:insert`	Inserts content into a template.
`ui:param`	Used to pass parameters to an included file.
`ui:repeat`	Used as an alternative for loop tags, such as `c:forEach` or `h:dataTable`.
`ui:remove`	Removes content from a page.

For more information on Facelets templating tags, see the documentation at http://download.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/.

The Facelets tag library includes the main templating tag `ui:insert`.

A template page that is created with this tag allows you to define a default structure for a page.

A template page is used as a template for other pages, usually referred to as client pages.

Here is an example of a template saved as `template.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:ui=
"http://java.sun.com/jsf/facelets"
```

```
xmlns:h=
"http://java.sun.com/jsf/html"
>
<h:head>
<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8"
/>
<link
href="./resources/css/default.css"
rel="stylesheet" type="text/css"
/>
```

```
<link
href=
"./resources/css/cssLayout.css"
rel="stylesheet" type="text/css"
/>
<title>Facelets Template</title>
</h:head>
<h:body>
<div id="top" class="top">
```

```
<ui:insert name="top">
Top Section
</ui:insert>
</div>
<div>
<div id="left">
<ui:insert name="left">
Left Section
</ui:insert>
</div>
```

```
<div id="content"
class="left_content"
>
<ui:insert name="content">
Main Content
</ui:insert>
</div>
</div>
</h:body>
</html>
```

The example page defines an XHTML page that is divided into three sections: a top section, a left section, and a main section.

The sections have style sheets associated with them.

The same structure can be reused for the other pages of the application.

The client page invokes the template by using the `ui:composition` tag.

In the following example, a client page named `templateclient.xhtml` invokes the template page named `template.xhtml` from the preceding example.

A client page allows content to be inserted with the help of the `ui:define` tag.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd"
>
<html
xmlns=
"http://www.w3.org/1999/xhtml"
xmlns:ui=
"http://java.sun.com/jsf/facelets"
```

```
xmlns:h=
"http://java.sun.com/jsf/html"
>
<h:body>
<ui:composition
template="./template.xhtml"
>
<ui:define name="top">
Welcome to Template Client Page
</ui:define>
```

```
<ui:define name="left">
<h:outputLabel
value="You are in the Left Section"
/>
</ui:define>
<ui:define name="content">
<h:graphicImage
value=
"#{resource['images:wave.med.gif']}"
/>
```

```
<h:outputText
value=
"You are in the Main Content Section"
/>
</ui:define>
</ui:composition>
</h:body>
</html>
```

You can use NetBeans IDE to create Facelets template and client pages.

For more information on creating these pages, see

http://netbeans.org/kb/docs/web/jsf20-intro.html.

Composite Components

JavaServer Faces technology offers the concept of composite components with Facelets.

A composite component is a special type of template that acts as a component.

Any component is essentially a piece of reusable code that behaves in a particular way.

For example, an `inputText` component accepts user input.

A component can also have validators, converters, and listeners attached to it to perform certain defined actions.

A composite component consists of a collection of markup tags and other existing components.

This reusable, user-created component has a customized, defined functionality and can have validators, converters, and listeners attached to it like any other component.

With Facelets, any XHTML page that contains markup tags and other components can be converted into a composite component.

Using the resources facility, the composite component can be stored in a library that is available to the application from the defined resources location.

Table 5-3 lists the most commonly used composite tags and their functions.

Table 5-3 Composite Component Tags

Tag	Function
`composite:interface`	Declares the usage contract for a composite component. The composite component can be used as a single component whose feature set is the union of the features declared in the usage contract.

`composite:`
`implementation`
	Defines the implementation of the composite component.
	
	If a `composite:interface` element appears, there must be a corresponding `composite:implementation`.

`composite:attribute`	Declares an attribute that may be given to an instance of the composite component in which this tag is declared.
`composite:insertChildren`	Any child components or template text within the composite component tag in the using page will be reparented into the composite component at the point indicated by this tag's placement within the `composite:implementation` section.
`composite:valueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ValueHolder` suitable for use as the target of attached objects in the using page.

`composite:editableValueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `EditableValueHolder` suitable for use as the target of attached objects in the using page.
`composite:actionSource`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ActionSource2` suitable for use as the target of attached objects in the using page.

For more information and a complete list of Facelets composite tags, see the documentation at http://download.oracle.com/javaee/6/javaserverfaces/2.1/docs/vdldocs/facelets/.

The following example shows a composite component that accepts an email address as input:

```
<!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:composite=
"http://java.sun.com/jsf/composite"
```

```
xmlns:h=
"http://java.sun.com/jsf/html"
>
<h:head>
<title>
This content will not be displayed
</title>
</h:head>
<h:body>
<composite:interface>
```

```
<composite:attribute
name="value" required="false"
/>
</composite:interface>
<composite:implementation>
<h:outputLabel value="Email id: ">
</h:outputLabel>
<h:inputText
value="#{cc.attrs.value}">
</h:inputText>
</composite:implementation>
```

```
</h:body>
</html>
```

Note the use of `cc.attrs.value` when defining the value of the `inputText` component.

The word `cc` in JavaServer Faces is a reserved word for composite components.

The `#{cc.attrs.`*`attribute-name`*`}` expression is used to access the attributes defined for the composite component's interface, which in this case happens to be `value`.

The preceding example content is stored as a file named `email.xhtml` in a folder named `resources/emcomp`, under the application web root directory.

This directory is considered a library by JavaServer Faces, and a component can be accessed from such a library.

For more information on resources, see Resources.

The web page that uses this composite component is generally called a using page.

The using page includes a reference to the composite component, in the `xml` namespace declarations:

```
<!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:em="http://java.sun.com/jsf/
composite/emcomp/"
>
<h:head>
```

```
<title>
Using a sample composite component
</title>
</h:head>
<body>
<h:form>
<em:email
value="Enter your email id"
/>
</h:form>
</body></html>
```

The local composite component library is defined in the `xml` namespace with the declaration `xmlns:em="http://java.sun.com/jsf/composite/emcomp/"`.

The component itself is accessed through the use of `em:email` tag.

The preceding example content can be stored as a web page named `emuserpage.xhtml` under the web root directory.

When compiled and deployed on a server, it can be accessed with the following URL:

```
http://localhost:8080/
application-name/
faces/emuserpage.xhtml
```

Resources

Web resources are any software artifacts that the web application requires for proper rendering, including images, script files, and any user-created component libraries.

Resources must be collected in a standard location, which can be one of the following.

- **A resource packaged in the web application root must be in a subdirectory of a `resources` directory at the web application root: `resources/`*`resource-identifier`*.**
- **A resource packaged in the web application's classpath must be in a subdirectory of the `META-INF/resources` directory within a web application: `META-INF/resources/`*`resource-identifier`*.**

The JavaServer Faces runtime will look for the resources in the preceding listed locations, in that order.

Resource identifiers are unique strings that conform to the following format:

```
[locale-prefix/]
[library-name/]
[library-version/]
resource-name[/resource-version]
```

Elements of the resource identifier in brackets (`[]`) are optional, indicating that only a ***`resource-name`***, which is usually a file name, is a required element.

Resources can be considered as a library location.

Any artifact, such as a composite component or a template that is stored in the `resources` directory, becomes accessible to the other application components, which can use it to create a resource instance.