JSF 2: Building Composite Components – Part I: Basics

JSF 2.2 Version

Topics in This Section

- **Idea**
- **Basic components**
- **Passing values to components**
- **Using ui:repeat inside components**
- **Handling relative URLs in components**
  - Hypertext links
  - Images
  - Style sheets
  - Scripts

Overview

Idea

- **JSF 2.0 added "composite components"**
  - Reusable, named chunks of facelets code
  - Can pass arguments
- **Similar to JSP tag files**
  - But have use of JSF tags and JSF expression language
  - Can have listeners, events, facets, etc.
- **Abbreviated example**
  - resources/utils/myComponent.xhtml (snippet)

```
<cc:implementation>
   This is a test
</cc:implementation>
```

  - some-page.xhtml (main page)

```
<utils:myComponent/>
```

Basic Components

Super-Quick Example

- **Component File**

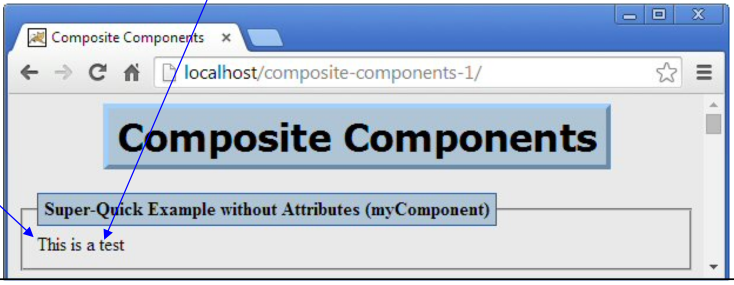
resources/utils/myComponent.xhtml

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:cc="http://xmlns.jcp.org/jsf/composite">

<cc:interface/>

<cc:implementation>
 This is a test
</cc:implementation>

</html>
```

- **Main File**

index.xhtml

```
<!DOCTYPE ...>
<html ...
   xmlns:utils=
      "http://xmlns.jcp.org/jsf/composite/utils">
<h:head><title>Composite Components</title>
...
<h:body>...
<utils:myComponent/>...
</h:body></html>
```


Composite Components
localhost/composite-components-1/
Composite Components
Super-Quick Example without Attributes (myComponent)
This is a test

Component File: Outline (WebContent/resources/someDir/someFile.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:cc="http://xmlns.jcp.org/jsf/composite">

<cc:interface>
  <!-- Declare attributes here -->
</cc:interface>

<cc:implementation>
  <!-- Create output here -->
</cc:implementation>

</html>
```

This component file must be in a folder under the "resources" folder, and the "resources" name is never referred to later, but is instead automatically assumed.

In the early days of JSF 2, it was common to use "composite" instead of "cc" as the namespace, resulting in composite:interface and composite:implementation instead of cc:interface and cc:implementation. This is still legal, of course, since you can define any namespace that you want. But, it is now slightly more customary to use "cc" as the namespace.

Main File: Outline

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:someDir="http://xmlns.jcp.org/jsf/composite/someDir">
<h:head>
…
</h:head>
<h:body>
…
<someDir:someFile/>
…
</h:body></html>
```

Technically, it is legal to use any namespace that you want. For example:
xmlns:blah="http://xmlns.jcp.org/jsf/composite/someDir"
You would then use that namespace in the main page: <blah:someFile/>. However, it is moderately conventional to choose a folder name under resources that looks readable as a namespace, and then to use that folder name as the namespace as above.

Longer Example

- **Make a bulleted list of request info**
  - Requesting host
  - User agent
  - Current Locale
  - Session ID
- **May want to reuse list, so make component**
  - resources/utils/info1.xhtml
    - Generates list inside cc:implementation
  - index.xhtml
    - Declares utils: namespace
    - Uses <utils:info1/>

Example: Summary

- **Component File**

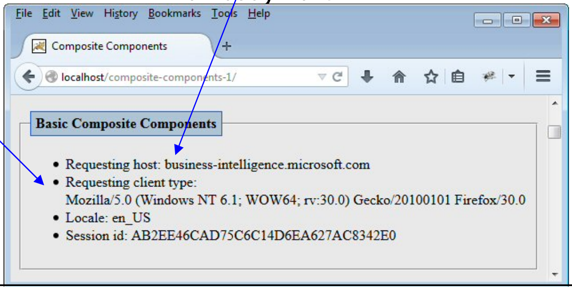
resources/utils/info1.xhtml

```
<!DOCTYPE ...>
<html ...
   xmlns:cc="http://xmlns.jcp.org/jsf/composite">

<cc:interface/>

<cc:implementation>
  <ul>
    <li>Requesting host: #{request.remoteHost}</li>
    <li>Requesting client type:<br/>
        #{header["User-Agent"]}</li>
    <li>Locale: #{view.locale}</li>
    <li>Session id: #{session.id}</li>
  </ul>
</cc:implementation>

</html>
```

- **Main File**

some-page.xhtml

```
<!DOCTYPE ...>
<html ...
   xmlns:utils=
     "http://xmlns.jcp.org/jsf/composite/utils">
<h:head><title>Composite Components</title>
...
<h:body>...
<utils:info1/>...
</h:body></html>
```


File Edit View History Bookmarks Tools Help
Composite Components
localhost/composite-components-1/

Basic Composite Components

- Requesting host: business-intelligence.microsoft.com
- Requesting client type:
  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0
- Locale: en_US
- Session id: AB2EE46CAD75C6C14D6EA627AC8342E0

Components with Attributes

Summary

- **Component file**
  - Same basic structure as before, in folder under resources
    - Declares cc: namespace as before
  - cc:interface defines attributes

```
<cc:interface>
   <cc:attribute name="attributeName"/>
</cc:interface>
```

  - cc:implementation outputs attributes via #{cc.attrs…}

```
<cc:implementation>
…#{cc.attrs.attributeName}…
</cc:implementation>
```

- **Main page**

```
<someDir:someFile attributeName="…"/>
```

More on cc:attribute

- **Basic usage**

```
<cc:interface>
   <cc:attribute name="attributeName"/>
</cc:interface>
```

- **Basic attributes**
  - name
    - Attribute name as used in component in main page
  - required
    - Is attribute required? (default: false)
  - default
    - For non-required attributes, value to use if none supplied
- **Advanced attributes**
  - type, method-signature, targets
    - For attributes that expect methods or complex values
    - The “type” attribute is especially useful; see examples in later composite component tutorials

More on cc.attrs

- **"cc" is predefined variable**
  - Stands for "composite component"
    - The *namespace* is "cc" by convention, but "composite" or other things are legal. But the cc *variable* name is predefined and cannot be changed.
  - Refers to top-level component in component file, of type UINamingContainer
- **Main property is "attrs"**
  - This is a specially defined attribute that contains a Map of the attributes used to call the component
    - Main page: **<someDir:someFile message="test"/>**
    - Component: **<h2>Message: #{cc.attrs.message}</h2>**
- **Other useful attributes**
  - parent, children, clientId

Mini Example (resources/utils/showMessages.xhtml)

Note: "required" was not properly enforced in Mojarra 2.0.2 and earlier.

- **cc:interface**

```
<cc:attribute name="msg1" required="true"/>
<cc:attribute name="msg2" default="Hi"/>
```

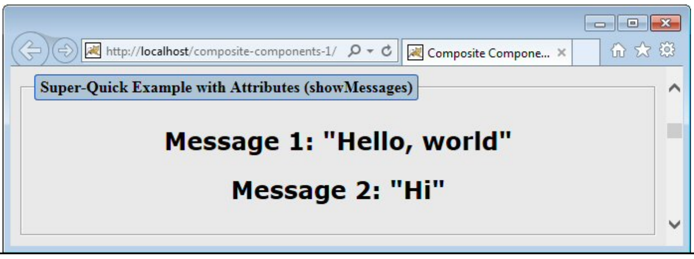
- **cc:implementation**

```
<h2>Message 1: "#{cc.attrs.msg1}"</h2>
<h2>Message 2: "#{cc.attrs.msg2}"</h2>
```

- **Main page (index.xhtml)**

```
<utils:showMessages msg1="Hello, world"/>
```

- **Result**



Super-Quick Example with Attributes (showMessages)

Message 1: "Hello, world"

Message 2: "Hi"

Longer Example

- **Make a bulleted list of request info as before**
  - Requesting host, user agent, current Locale, session ID
- **Support styleClass attribute**
  - To specify CSS style of resultant <ul> list
- **Component**
  - resources/utils/info2.xhtml
    - Declares styleClass inside cc:interface
    - Generates list inside cc:implementation
  - index.xhtml
    - Declares utils: namespace
    - Uses <utils:info2 styleClass="some-css-name"/>

Example: Summary

- **Component file: cc:interface**

```
<cc:attribute name="styleClass"/>
```

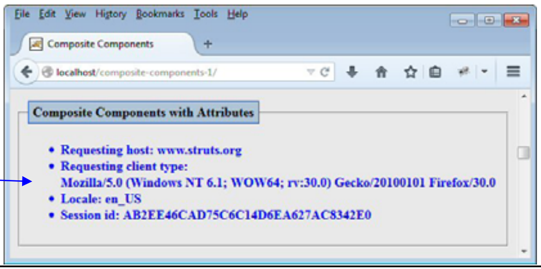
- **Component file: cc:implementation**

```
<ul class="#{cc.attrs.styleClass}">
  <li>Requesting host: #{request.remoteHost}</li>
  <li>Requesting client type<br/> #{header["User-Agent"]}</li>
  <li>Locale: #{view.locale}</li>
  <li>Session id: #{session.id}</li>
 </ul>
```

- **Main page**

```
<utils:info2 styleClass="blue"/>
```

- **Result**


File Edit View History Bookmarks Tools Help
Composite Components
localhost/composite-components-1/

Composite Components with Attributes

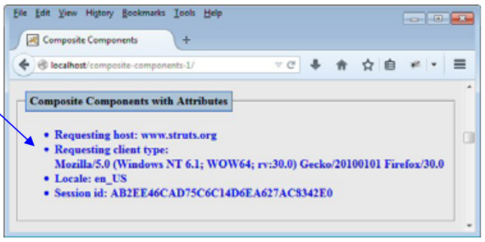
- Requesting host: www.struts.org
- Requesting client type:
  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0
- Locale: en_US
- Session id: AB2EE46CAD75C6C14D6EA627AC8342E0

Component File: Details (resources/utils/info2.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:cc="http://xmlns.jcp.org/jsf/composite">

<cc:interface>
  <cc:attribute name="styleClass"/>
</cc:interface>

<cc:implementation>
  <ul class="#{cc.attrs.styleClass}">
    <li>Requesting host: #{request.remoteHost}</li>
    <li>Requesting client type:
        <br/> #{header["User-Agent"]}</li>
    <li>Locale: #{view.locale}</li>
    <li>Session id: #{session.id}</li>
  </ul>
</cc:implementation>
</html>
```

Main File: Details (index.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:utils="http://xmlns.jcp.org/jsf/composite/utils">
<h:head>
…
</h:head>
<h:body>
…
<fieldset>
<legend>Composite Components with Attributes</legend>
<utils:info2 styleClass="blue"/>
</fieldset>
…
</h:body></html>
```


File Edit View History Bookmarks Tools Help
Composite Components
localhost/composite-components-1/

Composite Components with Attributes

- Requesting host: www.struts.org
- Requesting client type:
  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:30.0) Gecko/20100101 Firefox/30.0
- Locale: en_US
- Session id: AB2EE46CAD75C6C14D6EA627AC8342E0

Using ui:repeat with Composite Components

Component File: Summary

- **In subfolder of "resources" folder as before**
  - E.g., resources/someDir/someFile.xhtml
- **Basic structure**
  - Looks like xhtml file as before
    - Declare cc: *and* ui: namespaces
      - <html … xmlns:cc="http://xmlns.jcp.org/jsf/composite"
        xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  - Contains cc:interface
    - Defines attributes as before
  - Contains cc:implementation
    - Creates final output
      - Uses #{cc.attrs.attributeName} as before
      - Uses ui:repeat to build output

More on ui:repeat

- **Sample usage**

```
<ul>
<ui:repeat var="color" value="#{item.availableColors}">
  <li>#{color}</li>
</ui:repeat>
</ul>
```

- **More attributes**

```
<ui:repeat var="someVar"
           value="#{someBean.someCollection}"
           varStatus="statusVariable"
           offset="…"
           size="…"
           step="…">
  …
</ui:repeat>
```

- **More details on ui:repeat**
  – See tutorial section on looping

Example

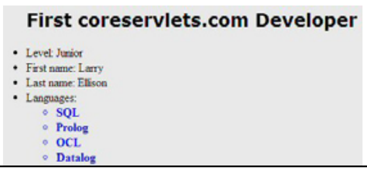
- **Pass in array or List**
  – Turn it into <ul> list showing individual entries
- **Attributes**
  – value (EL expression to designate collection)
  – styleClass (to specify CSS style of resultant <ul> list)
- **Component**
  – resources/utils/list.xhtml
    - Declares value and styleClass inside cc:interface
    - Generates list using ui:repeat inside cc:implementation
  – index.xhtml
    - Declares utils: namespace
    - Uses <utils:list value="#{someBean.someCollection}"
      styleClass="some-css-name"/>

Component File (resources/utils/list.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:cc="http://xmlns.jcp.org/jsf/composite"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<cc:interface>
  <cc:attribute name="value" required="true"/>
  <cc:attribute name="styleClass" default="italic"/>
</cc:interface>

<cc:implementation>
  <ul class="#{cc.attrs.styleClass}">
  <ui:repeat var="listItem" value="#{cc.attrs.value}">
    <li>#{listItem}</li>
  </ui:repeat>
  </ul>
</cc:implementation>
</html>
```

Main File (index.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:utils="http://xmlns.jcp.org/jsf/composite/utils">
…
<fieldset>
<legend>Composite Components that Use ui:repeat</legend>
<h2>First coreservlets.com Developer</h2>
<ul>
  <li>Level: #{person1.level}</li>
  <li>First name: #{person1.firstName}</li>
  <li>Last name: #{person1.lastName}</li>
  <li>Languages: <utils:list value="#{person1.languages}"
                              styleClass="blue"/></li>
</ul>
```

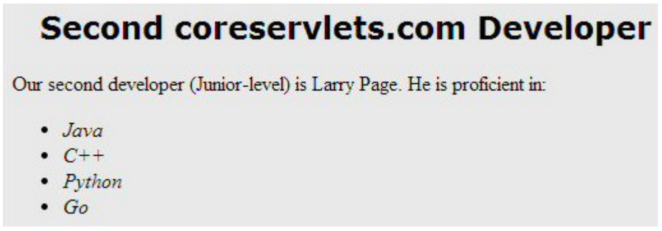


First coreservlets.com Developer

- Level: Junior
- First name: Larry
- Last name: Ellison
- Languages:
  - **SQL**
  - **Prolog**
  - **OCL**
  - **Datalog**

Main File: Continued (index.xhtml)

```
<hr/>
<h2>Second coreservlets.com Developer</h2>
<p>
Our second developer (#{person2.level}-level) is
#{person2.firstName} #{person2.lastName}.
He is proficient in:</p>
<utils:list value="#{person2.languages}"/>
</fieldset>
<p/>
<fieldset>
```



Second coreservlets.com Developer

Our second developer (Junior-level) is Larry Page. He is proficient in:

- *Java*
- *C++*
- *Python*
- *Go*

Base Class for Beans

```
public class Programmer {
  private String firstName, lastName, level;
  private String[] languages;

  public Programmer(String firstName,
                    String lastName,
                    String level,
                    String... languages) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.level = level;
    this.languages = languages;
  }

  // Getter methods getFirstName, getLastName, etc.
  // Note that getLanguages returns an array
```

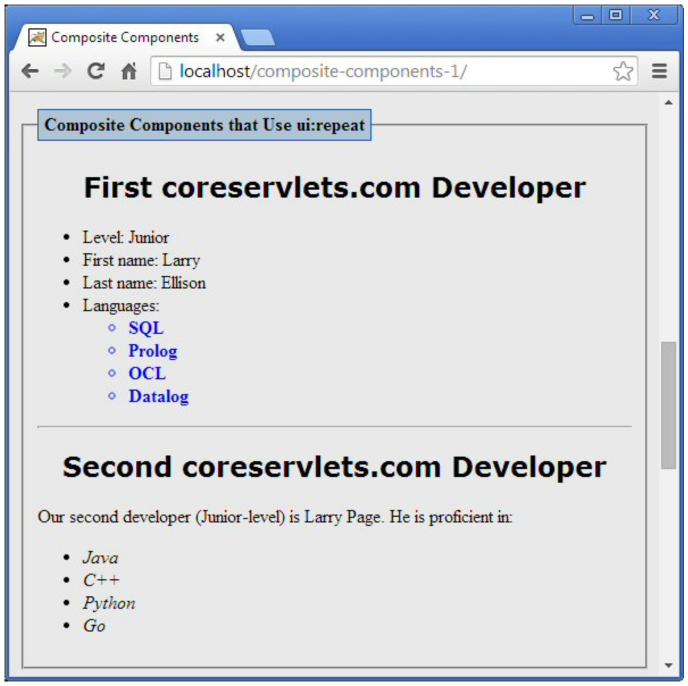
Bean: Person1

```
@ManagedBean
public class Person1 extends Programmer {
  public Person1() {
    super("Larry", "Ellison", "Junior",
          "SQL", "Prolog", "OCL", "Datalog");
  }
}
```

Bean: Person2

```
@ManagedBean
public class Person2 extends Programmer {
  public Person2() {
    super("Larry", "Page", "Junior",
          "Java", "C++", "Python", "Go");
  }
}
```

Results



Composite Components that Use ui:repeat

First coreservlets.com Developer

- Level: Junior
- First name: Larry
- Last name: Ellison
- Languages:
  - SQL
  - Prolog
  - OCL
  - Datalog

Second coreservlets.com Developer

Our second developer (Junior-level) is Larry Page. He is proficient in:

- *Java*
- *C++*
- *Python*
- *Go*

Handling Relative URLs in Components

Problems with Simple Relative URLs

- **Issue**
  - Suppose component used simple relative URLs
  - Suppose main page was moved to subdirectory
- **Example**
  - Hypertext links
    - <a href="welcome.jsf"> would now refer to http://host/context-root/**subdir**/welcome.jsf
  - Images
    - <img src="./images/pic.jpg"> would now refer to http://host/context-root/**subdir**/images/pic.jpg
  - Style sheets
    - <link …href="./css/styles.css"/> would now refer to http://host/context-root/**subdir**/css/styles.css
  - JavaScript files
    - <script src="./scripts/my-script.js"…> would now refer to http://host/context-root/**subdir**/scripts/my-script.js

Solutions (Relocatable Resources)

- **Hypertext links**
  - Use <a href="#{request.contextPath}/blah.jsf">
    - Or <h:outputLink value="#{request.contextPath}/blah.jsf"/>
- **Images**
  - Put in images folder under "resources" and use <h:graphicImage name="blah.jpg" library="images"/>
    - Can also use <h:graphicImage url="/images/blah.jpg"/> for context-independent ref to images *outside* resources folder
- **Style sheets**
  - Put in css folder under "resources" and use <h:outputStylesheet name="blah.css" library="css"/>
- **JavaScript files**
  - Put in scripts folder under "resources" and use <h:outputScript name="blah.js" library="scripts" target="head"/>

Reminder – h:head and h:body

- **Plan ahead: always use h:head and h:body**
  - Never just <head> and <body> (so JSF can find regions)
- **Reasons**
  - f:ajax
    - This tag automatically inserts scripts, and can't find the place to insert unless you use h:head and h:body
  - h:outputStylesheet
    - <h:outputStylesheet> does not have to be in the head, so JSF needs to be able to find the head to insert the final output (<link>) that loads the style sheet.
  - h:outputScript
    - Again, <h:outputScript> does not have to be in the head, so JSF needs to be able to find the head (if you use target="head") or body (if you use target="body") to insert the final output (<script>) that loads the JavaScript file

Hypertext Links: Relative URLs

- **Prepend #{request.contextPath}**
  - Unlike images, style sheets, and scripts, JSF has no builtin way to build context-relative links.
  - However, it is easy to do so yourself
- **Examples**
  - <a href="#{request.contextPath}/blah.jsf">
  - <h:outputLink value="#{request.contextPath}/blah.jsf"/>
    - Assume that your context root is /my-context. Then, both of the above build <a href="/my-context/blah.jsf">

Images: Relative URLs

- **Use h:graphicImage with name and library**
  - JSF 2.0 added the concept of resource folders. You make a folder literally called “resources”, then make subfolders inside (you can choose the name of the folders inside).
    - For “library”, give the name of the images subfolder
    - For “name”, give the image filename
  - This has two advantages beyond just context-relative URLs: versioning, localization, and resources in the classpath
  - Or, use <h:graphicImage url="/foo/bar.jpg"/>
    - Outputs <img src="/my-context/foo/bar.jgp"/>. This is a holdover from JSF 1.x. Resources are better in most cases.
- **Example**
  - <h:graphicImage name="blah.gif" library="images"/>
    - Outputs <img src="…"/>, where src refers to JSF page that outputs /my-context/resources/images/blah.gif

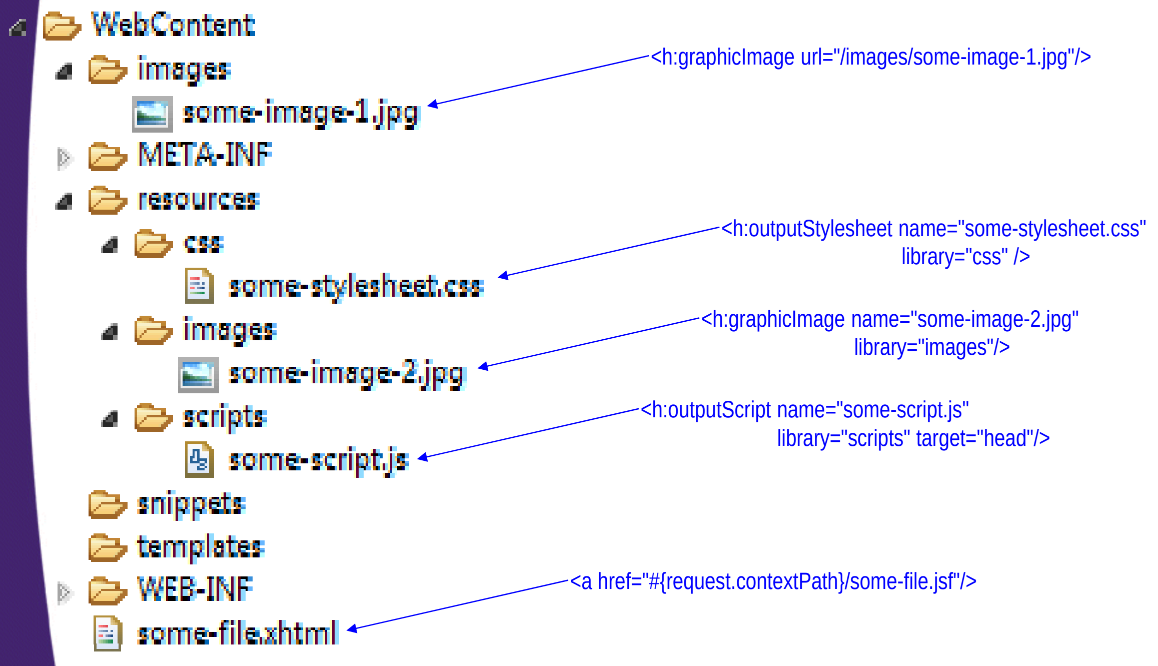
Style Sheets: Relative URLs

- **Use h:outputStylesheet with name & library**
  - Again, make “resources” folder with subfolder inside
    - For “library”, give the name of the CSS subfolder
    - For “name”, give the filename of the stylesheet
  - It is not necessary to put <h:outputStylesheet> in the head
    - This makes it easy for components or included pages to refer to the style sheets they need. The real reference will come out in the head, and if there are multiple uses of h:outputStylesheet in the same page (e.g., from two included pages), it will build only one ref in main page
- **Example**
  - <h:outputStylesheet name="blah.css" library="css"/>
    - Outputs <link type="text/css" rel="stylesheet" href="…" /> where href refers to JSF page that outputs /my-context/resources/css/blah.css

Scripts: Relative URLs

- **Use h:outputScript with name & library**
  - Again, make “resources” folder with subfolder inside
    - For “library”, give the name of the JavaScript subfolder
    - For “name”, give the name of the JavaScript file
  - It is not necessary to put <h:outputScript> in the head
    - As with style sheets, this makes it easy for components or included pages to refer to the scripts they need. Again, if the tag is used multiple times, only one ref to the script appears in the real page.
    - If you omit target="head“, script appears at current location instead of in head. body and form are also legal, but rare, options.
- **Example**
  - <h:outputScript name="blah.js" library="scripts" target="head" />
    - Outputs <script type="text/javascript" src="…" /> in head, where src refers to page that outputs /my-context/resources/scripts/blah.js

Example: Referring to Files from Templates or Components



<h:graphicImage url="/images/some-image-1.jpg"/>

<h:outputStylesheet name="some-stylesheet.css" library="css" />

<h:graphicImage name="some-image-2.jpg" library="images"/>

<h:outputScript name="some-script.js" library="scripts" target="head"/>

<a href="#{request.contextPath}/some-file.jsf"/>

Component File (resources/utils/info3.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html … xmlns:cc="http://xmlns.jcp.org/jsf/composite"
      xmlns:utils="http://xmlns.jcp.org/jsf/composite/utils">
…
<cc:interface>
  <cc:attribute name="styleClass"/>
</cc:interface>
<cc:implementation>
  <table align="right">
    <tr><th>
      <h:graphicImage name="duke-guitar.png" library="images"/>
    </th></tr>
    <tr><th>
      <a href="#{request.contextPath}/index.jsf">
      Return to main page</a>
    </th></tr>
  </table>
  <utils:info2 styleClass="#{cc.attrs.styleClass}"/>
</cc:implementation>
</body></html>
```

Actual file is resources/images/duke-guitar.png

Main File 1 (relative-urls.xhtml)

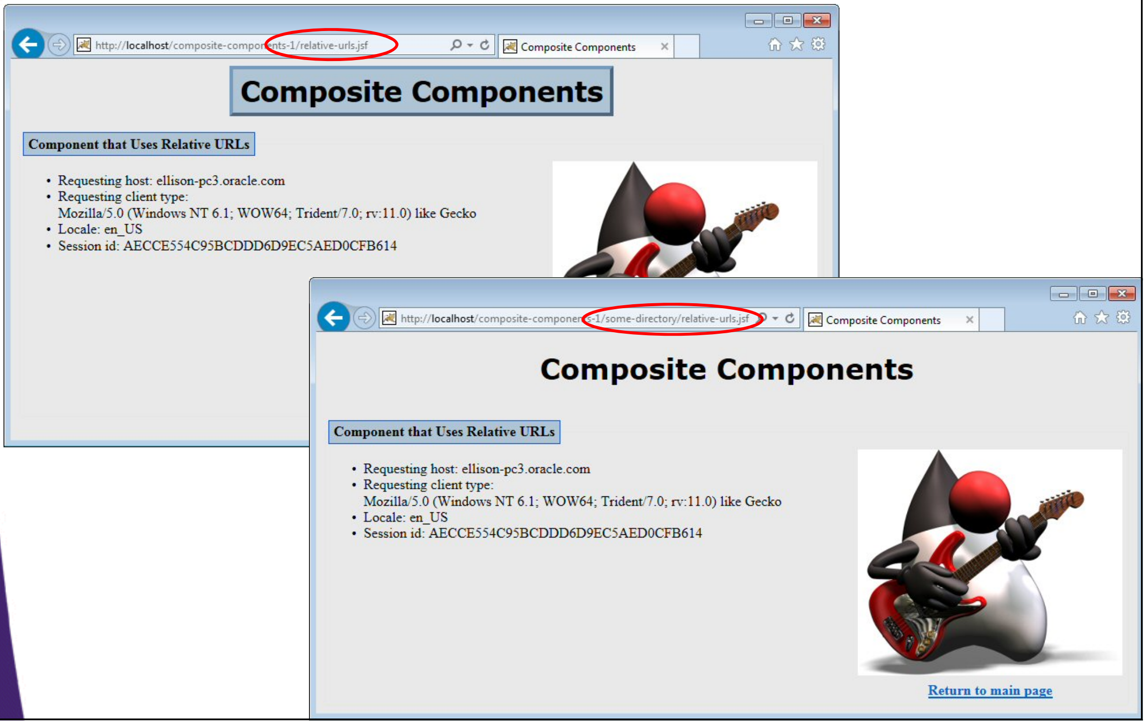
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:utils="http://xmlns.jcp.org/jsf/composite/utils">
<h:head><title>Composite Components</title>
<h:outputStylesheet name="styles.css" library="css"/>
</h:head>
<h:body>
…
<fieldset>
<legend>Component that Uses Relative URLs</legend>
<utils:info3/>
</fieldset>
…
</h:body></html>
```

Main File 2 (someDirectory/relative-urls.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:utils="http://xmlns.jcp.org/jsf/composite/utils">
<h:head><title>Composite Components</title>
<h:outputStylesheet name="styles.css" library="css"/>
</h:head>
<h:body>
…
<fieldset>
<legend>Component that Uses Relative URLs</legend>
<utils:info3/>
</fieldset>
…
</h:body></html>
```

This is an exact copy of the file from the top-level directory.

Results


http://localhost/composite-components-1/relative-urls.jsf
Composite Components
http://localhost/composite-components-1/some-directory/relative-urls.jsf
Composite Components

Composite Components

Component that Uses Relative URLs

- Requesting host: ellison-pc3.oracle.com
- Requesting client type: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
- Locale: en_US
- Session id: AECCE554C95BCDDD6D9EC5AED0CFB614

Composite Components

Component that Uses Relative URLs

- Requesting host: ellison-pc3.oracle.com
- Requesting client type: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
- Locale: en_US
- Session id: AECCE554C95BCDDD6D9EC5AED0CFB614

Return to main page

Wrap-Up

Summary

- **Define a component**
  - Put foo.xhtml in "resources/bar" folder
  - Define available attributes inside **cc:interface**
    - Use <cc:attribute name="attributeName" … />
  - Build output inside **cc:implementation**
    - Output attributes with #{cc.attrs.attributeName}
    - Use ui:repeat, f:ajax, h:dataTable, & other JSF constructs
    - Be careful with relative URLs.
- **Use component in main page**
  - Define namespace
    - <html …
      xmlns:bar="http://xmlns.jcp.org/jsf/composite/bar">
  - Use the component in facelets page
    - <bar:foo attributeName="…" />

Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training