10

Facelets and its use in Web Applications

As of version 2 of this specification, JavaServer Faces implementations must support (although JSF-based applications need not utilize) using Facelets as the view declaration language for JSF pages. Facelets technology was created by JSR-252 EG Member Jacob Hookom.

10.1 Non-normative Background

To aid implementors in providing a spec compliant runtime for Facelets, this section provides a non-normative background to motivate the discussion of the Facelets feature. Facelets is a replacement for JSP that was designed from the outset with JSF in mind. New features introduced in version 2 and later are only exposed to page authors using Facelets. JSP is retained for backwards compatibility.

10.1.1 Differences between JSP and Facelets

Facelets was the first non-JSP view declaration language designed for JavaServer Faces. As such, Facelets was able to provide a simpler and more powerful programming model to JSF developers than that provided by JSP, largely by leveraging JSF as much as possible without carrying backwards compatibility with JSP. The following table lists some of the differences between Facelets and JSP

TABLE 10-1 Comparison of Facelets and JSP

Feature Name	JSP	Facelets
Pages are compiled to...	A Servlet that gets executed each time the page renders. The UIComponent hierarchy is built by the presence of custom tags in the page.	An abstract syntax tree that, when executed, builds a UIComponent hierarchy.
Handling of tag attributes	All tag attributes must be declared in a TLD file. Conformance instances of components in a page with the expected attributes can be enforced with a taglibrary validator.	Tag attributes are completely dynamic and automatically map to properties, attributes and ValueExpressions on UIComponent instances
Page templating	Not supported, must go outside of core JSP	Page templating is a core feature of Facelets

TABLE 10-1 Comparison of Facelets and JSP

Feature Name	JSP	Facelets
Performance	Due to the common implementation technique of compiling a JSP page to a Servlet, performance can be slow	Facelets is simpler and faster than JSP
EL Expressions	Expressions in template text cause unexpected behavior when used in JSP	Expressions in template text operate as expected.
JCP Standard	Yes, the specification is separate from the implementation for JSP	No, the specification is defined by and is one with the implementation.

10.1.2 Differences between Pre JSF 2.0 Facelets and Facelets in JSF 2.0

The work of taking a snapshot of a version of Facelets and producing the specification for Facelets in JSF 2.0 consists of extracting the parts of Facelets that are intended to be "public" and leaving the rest as implementation details. A decision was made early in this process to strive for backwards compatibility between the latest popular version of Facelets and Facelets in JSF 2.0. The sole determinant to backwards compatibility lies in the answer to the question, "is there any Java code in the application, or in libraries used by the application, that extends from or depends on any class in package `com.sun.facelets` and/or its sub-packages?"

If the answer to this question is "yes", Facelets in JSF 2.0 is *not* backwards compatibile with Facelets and such an application *must* continue to bundle the Facelets jar file along with the application, continue to set the Facelets configuration parameters, and also set the `javax.faces.DISABLE_FACELET_JSF_VIEWHANDLER` `<context-param>` to `true`. Please see Section 11.1.3 "Application Configuration Parameters" for details on this option. Any code that extends or depends on any class in package `com.sun.facelets` and/or its sub-packages must be modified to depend on the appropriate classes in package `javax.faces.webapp.vdl` and/or its sub-packages.

If the answer to this question is "no", Facelets in JSF 2.0 *is* backwards compatible with pre-JSF 2.0 Facelets and such an application *must not* continue to bundle the Facelets jar file along with the application, and *must not* continue to set the Facelets configuration parameters.

Thankfully, most applications that use Facelets fall into the latter category, or, if they fall in the former, their dependence will easily be migrated to the new public classes.

Facelets in JSF 2.0 provides tag libraries that are compatible with the following libraries already found in pre JSF 2.0 Facelets.

TABLE 10-2 Taglibs in pre JSF 2.0 Facelets that are available in Facelets in JSF 2.0

Common prefix	Namespace URI
h	http://java.sun.com/jsf/html
f	http://java.sun.com/jsf/core
c	http://java.sun.com/jsp/jstl/core
fn	http://java.sun.com/jsp/jstl/functions
ui	http://java.sun.com/jsf/facelets

Naturally, new features built on Facelets in JSF 2.0 are not available in pre JSF 2.0 Facelets and will only work in JSF 2.0 or later.

10.2 Java Programming Language Specification for Facelets in JSF 2.0

The subsections within this section specify the Java API requirements of a Facelets implementation. Adherence to this section and the next section, which specifies the XHTML specification for Facelets in JSF 2.0, will ensure applications and JSF component libraries that make use of Facelets are portable across different implementations of JavaServer Faces.

The original Facelet project did not separate the API and the implementation into separate jars, as is common practice with JCP specifications. Thus, a significant task for integrating Facelets into JSF 2 was deciding which classes to include in the public Java API, and which to keep as an implementation detail.

There were two guiding principles that influenced the task of integrating Facelets into JSF 2.

The original decision in JSF 1.0 to allow the ViewHandler to be pluggable enabled the concept of a View Declaration Language for JSF. The two most popular ones were Facelets and JSFTemplating. The new integration should preserve this pluggability, since it is still valuable to be able to replace the View Declaration Language.

After polling users of Facelets, the expert group decided that most of them were only using the markup based API and were not extending from the Java classes provided by the Facelet project. Therefore, we decided to keep the Java API for Facelets in JSF 2 as small as possible, only exposing classes where absolutely necessary.

The application of these principles produced the classes in the package `javax.faces.view.facelets`. Please consult the Javadocs for that package, and the classes within it, for additional normative specification.

10.2.1 Specification of the ViewDeclarationLanguage Implementation for Facelets for JSF 2.0

As normatively specified in the javadocs for `ViewDeclarationLanguageFactory.getViewDeclarationLanguage()`, a JSF implementation must guarantee that a valid and functional `ViewDeclarationLanguage` instance is returned from this method when the argument is a refrence to either a JSP view, a Faces XML View or a Facelets View. This section describes the specification for the Facelets implementation.

```
public void buildView(FacesContext context,
                              UIViewRoot root)
                       throws IOException
```

The argument `root` will have been created with a call to either `createView()` or `ViewMetadata.createMetadataView()`. If the root already has non-metadata children, the view must still be re-built, but care must be taken to ensure that the existing components are correctly paired up with their VDL counterparts in the VDL page. The implementation must examine the `viewId` of the argument root, which must resolve to an entity written in Facelets for JSF 2 markup language. Because Facelets for JSF 2.0 views are written in XHTML, an XML parser is well suited to the task of processing such an entity. Each element in the XHTML view falls into one of the following categories, each of which corresponds to an instance of a Java object that implements `javax.faces.view.facelets.FaceletHandler`, or a subinterface or subclass thereof, and an instance of `javax.faces.view.facelets.TagConfig`, or a subinterface or subclass thereof, which is passed to the constructor of the object implementing `FaceletHandler`.

When constructing the `TagConfig` implementation to be passed to the `FaceletHandler` implementation, the runtime must ensure that the instance returned from `TagConfig.getTag()` has been passed through the tag decoration process as described in the javadocs for `javax.faces.view.facelets.TagDecorator` prior to the `TagConfig` being passed to the `FaceletHandler` implementation.

The mapping between the categories of elements in the XHTML view and the appropriate sub-interface or subclass of `FaceletHandler` is specified below. Each `FaceletHandler` instance must be traversed and its `apply()` method called in the same depth-first order as in the other lifecycle phase methods in jsf. Each `FaceletHandler` instance must use the `getNextHandler()` method of the `TagConfig` instance passed to its constructor to perform the traversal starting from the root `FaceletHandler`.

Standard XHTML markup elements

These are declared in the XHTML namespace `http://www.w3.org/1999/xhtml`. Such elements should be passed through as is to the rendered output.

These elements correspond to instances of `javax.faces.view.facelets.TextHandler`. See the javadocs for that class for the normative specification.

Markup elements that represent `UIComponent` instance in the view.

These elements can come from the Standard HTML Renderkit namespace `http://java.sun.com/jsf/html`, or from the namespace of a custom tag library (including composite components) as described in Section 10.3.2 “Facelet Tag Library mechanism”.

These elements correspond to instances of `javax.faces.view.facelets.ComponentHandler`. See the javadocs for that class for the normative specification.

Markup elements that take action on their parent or children markup element(s). Usually these come from the JSF Core namespace `http://java.sun.com/jsf/core`, but they can also be provided by a custom tag library.

Such elements that represent an attached object must correspond to an appropriate subclass of `javax.faces.view.facelets.FaceletsAttachedObjectHandler`. The supported subclasses are specified in the javadocs.

Such elements that represent a facet component must correspond to an instance of `javax.faces.component.FacetHandler`.

Such elements that represent an attribute that must be pushed into the parent `UIComponent` element must correspond to an instance of `javax.facelets.view.facelets.AttributeHandler`.

Markup Elements that indicate facelet templating, as specified in the VDL Docs for the namespace `http://java.sun.com/jsf/facelets`.

Such elements correspond to an instance of `javax.faces.view.facelets.TagHandler`.

Markup elements from the Facelet version of the JSTL namespaces `http://java.sun.com/jsp/jstl/core` or `http://java.sun.com/jsp/jstl/functions`, as specified in the VDL Docs for those namespaces.

Such elements correspond to an instance of `javax.faces.view.facelets.TagHandler`.

10.2.2 Resource Library Contracts

JSF defines a system called “resource library contracts” for applying facelet templates to an entire application in a re-usable and interchangeable manner. A customizable set of Facelet VDL views in the application will be able to declare themselves to be template-clients of any template in a resource library contract. Facelet VDL views in the application can also make use of resources contained in a resource library contract, but the feature has ample value when only used with templates.

10.2.2.1 Non-normative Example

Consider this resource library contract, called `siteLayout`.

```
siteLayout/
  topNav_template.xhtml
  leftNav_foo.html
  styles.css
  script.js
  background.png
```

The siteLayout contract offers two templates: topNav_template.xhtml and leftNav_foo.xhtml. For discussion, these are known as “declared templates”. When used by a template client, they will lay out the template client’s contents with a navigation menu on the top or the left side of the page, respectively. In siteLayout, each of the templates has <ui:insert> tags named “title”, “main”, and “footer”. For discussion, these are knows as “declared insertion points”. Furthermore, each of the templates uses the CSS styles declared in styles.css, some scripts defined in script.js, and the background image background.png. For discussion, these are known as “declared resources”. In order to use a resource library contract, one must know its declared templates, their declared insertion points, and, optionally, their declared resources. No constraint is placed on the naming and arrangement of declared templates, insertion points, or resources.

The resource library contract feature builds completely on top of the existing resource library concept introduced in JSF 2.0. The packaging, location, and localization requirements are identical to those in Section 2.6.1 “Packaging Resources” except that resource library contracts go in the contracts directory instead of the resources directory.

Once a resource library contract has been made available to the application, any Facelet VDL view in the application may be a template client to any of the declared templates in the contract, provided the declared insertion points are known. There are four ways to declare that a Facelet VDL view should be made to be a template client of one or more declared templates: 1. implicitly, by the runtime discovering the available resource library contracts and the VDL view author having a-priori knowledge of the declared templates, insertion points, and resources. 2.content in the application configuration resources declaring which views should be m. 3. the contracts attribute on <f:view> (and only on the **outer-most** <f:view>). 4. explicitly declaring a contract attribute on <ui:composition> or <ui:decorate>. Note that the first usage provides for the highest degree of interchangeability because no information about the name of the contract is included anywhere in the application. In this usage, any contract that has the same set of declared templates and insertion points (and resources, if those are to be considered as a part of the contract) may be used by the application. The first, implicit, usage will be carried through in the example.

The application includes siteLayout.jar in its WEB-INF/lib directory. Assume the contract is correctly packaged for auto-discovery. This application is very simple, it only has two pages: index.xhtml and page2.xhtml, as shown here.

index.xhtml.

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
<body>
<ui:composition template="/topNav_template.xhtml">
  <ui:define name="title">#{msgs.contactsWindowTitle}</ui:define>
  <ui:define name="main">
    <h:commandButton value="next" action="page2" />
  </ui:define>
  <ui:define name=
"footer">#{msgs.contactsWindowFooter}</ui:define>
</ui:composition>
</body>
</html>
```

page2.xhtml

```
<!DOCTYPE HTML>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
<body>
<ui:composition template="/leftNav_foo.xhtml">
  <ui:define name="main">
    <h:commandButton value="back" action="index" />
  </ui:define>
  <ui:define name=
"footer">#{msgs.contactsWindowFooter}</ui:define>
</ui:composition>
</body>
</html>
```

When the user agent visits `index.xhtml` because the `siteLayout` resource library contract provides `/topNav_template.xhtml`, that file will be loaded as the template. Likewise, when the `next` button is pressed, `/leftNav_foo.xhtml`, also from `siteLayout`,will be loaded as the template. In this use case, any resource library contract that declares templates and insertion points that match this usage is able to be used in place of the original `siteLayout`.

[P1-start_facelet_multi_template_inclusion]

10.2.2.2 Identifying the Multi-Template

PENDING: do we need to support a list of multi-templates that must be applied, or is one enough?

The multi-template to apply to the application is identified by means of the `javax.faces.MULTI_TEMPLATE` `<context-param>`. If such a parameter exists its value is taken to be the name of a resource library *multiTemplate*. Otherwise, the value of the symbolic constant `ResourceHandler.DEFAULT_MULTI_TEMPLATE` is taken to be the name of a resource library *multiTemplate*. If a library named *multiTemplate* is accessible to the application, and it contains a resource named `template.xhtml`, that resource is assumed to be suitable for use as the template for an implicit `<ui:composition>` that is made to be the template for every Facelet VDL view in the application. Otherwise, the multi-templating feature is effectively disabled for the application.

10.3 XHTML Specification for Facelets for JSF 2.0

10.3.1 General Requirements

[P1-start_facelet_xhtml]Facelet pages are authored in XHTML. The runtime must support all XHTML pages that conform to the XHTML-1.0-Transitional DTD, as described at `http://www.w3.org/TR/xhtml1/#a_dtd_XHTML-1.0-Transitional`.

The runtime must ensure that EL expressions that appear in the page without being the right-hand-side of a tag attribute are treated as if they appeared on the right-hand-side of the `value` attribute of an `<h:outputText />` element in the `http://java.sun.com/jsf/html` namespace. This behavior must happen regardless of whether or not the `http://java.sun.com/jsf/html` namespace has been declared in the page.

10.3.1.1 DOCTYPE and XML Declaration

When processing Facelet VDL files, the system must ensure that at most one XML declaration and at most one DOCTYPE declaration appear in the rendered markup, if and only if there is corresponding markup in the Facelet VDL files for those elements. If multiple occurrences of XML declaration and DOCTYPE declaration are encountered when processing Facelet VDL files, the "outer-most" occurrence is the one that must be rendered. If an XML declaration is present, it must be the very first markup rendered, and it must precede any DOCTYPE declaration (if present). The output of the XML and DOCTYPE declarations are subject to the configuration options listed in the table titled "Valid <process-as> values and their implications on the processing of Facelet VDL files" in Section 1.1.1.1 "The facelets-processing element".

[P1-end_facelet_xhtml]

10.3.2 Facelet Tag Library mechanism

Facelets leverages the XML namespace mechanism to support the concept of a "tag library" analogous to the same concept in JSP. However, in Facelets, the role of the tag handler java class is greatly reduced and in most cases is unnecessary. The tag library mechanism has two purposes.

Allow page authors to access tags declared in the supplied tag libraries declared in Section 10.4 "Standard Facelet Tag Libraries", as well as accessing third-party tag libraries developed by the application author, or any other third party

Define a framework for component authors to group a collection of custom `UIComponents` into a tag library and expose them to page authors for use in their pages.

[P1_start_facelet_taglib_decl]The runtime must support the following syntax for making the tags in a tag library available for use in a Facelet page.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:prefix="namespace_uri">
```

Where *`prefix`* is a page author chosen arbitrary string used in the markup inside the `<html>` tag to refer to the tags declared within the tag library and *`namespace_uri`* is the string declared in the `<namespace>` element of the facelet tag library descriptor. For example, declaring `xmlns:h="http://java.sun.com/jsf/html"` within the `<html>` element in a Facelet XHTML page would cause the runtime to make all tags declared in Section 10.4.2 "Standard HTML RenderKit Tag Library" to be available for use in the page using syntax like: `<h:inputText />`.

The unprefixed namespace, also known as the root namespace, must be passed through without modification or check for validity. The passing through of the root namespace must occur on any non-prefixed element in a facelet page. For example, the following markup declaration:.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
     <msup>
        <msqrt>
           <mrow>
              <mi>a</mi>
               <mo>+</mo>

               <mi>b</mi>
            </mrow>
         </msqrt>
      <mn>27</mn>
    </msup>
</math>
```

would be rendered as

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
     <msup>
        <msqrt>
           <mrow>
              <mi>a</mi>
               <mo>+</mo>

               <mi>b</mi>
            </mrow>
         </msqrt>
      <mn>27</mn>
    </msup>
</math>
```

[P1_end_facelet_taglib_decl]

[P1_start_facelet_taglib_discovery]The run time must support two modes of discovery for Facelet tag library descriptors

Via declaration in the web.xml, as specified in Section 11.1.3 “Application Configuration Parameters”

Via auto discovery by placing the tag library discriptor file within a jar on the web application classpath, naming the file so that it ends with “`.taglib.xml`”, without the quotes, and placing the file in the `META-INF` directory in the jar file.

The discovery of tag library files must happen at application startup time and complete before the application is placed in service. Failure to parse, process and otherwise interpret any of the tag library files discovered must cause the application to fail to deploy and must cause an informative error message to be logged.[P1_end_facelet_taglib_discovery]

The specification for how to interpret a facelet tag library descriptor is included in the documentation elements of the schema for such files, see Section 1.2 “XML Schema Definition For Facelet Taglib”.

10.3.3 Requirements specific to composite components

The text in this section makes use of the terms defined in Section 3.6.1.6 “Composite Component Terms”. When such a term appears in this section, it will be in *emphasis font face*.

10.3.3.1 Declaring a composite component library for use in a Facelet page

[P1_start_composite_library_decl]The runtime must support the following two ways of declaring a *composite component library*.

If a facelet taglibrary is declared in an XHTML page with a namespace starting with the string “`http://java.sun.com/jsf/composite/`” (without the quotes), the remainder of the namespace declaration is taken as the name of a resource library as described in Section 2.6.1.4 “Libraries of Localized and Versioned Resources”, as shown in the following example:

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

The runtime must look for a resource library named `ezcomp`. If the substring following “`http://java.sun.com/jsf/composite/`” contains a “/” character, or any characters not legal for a library name the following action must be taken. If `application.getProjectStage()` is `Development` an informative error message must be placed in the page and also logged. Otherwise the message must be logged only.

As specified in facelet taglibrary schema, the runtime must also support the `<composite-library-name>` element. The runtime must interpret the contents of this element as the name of a resource library as described in Section 2.6.1.4 “Libraries of Localized and Versioned Resources”. If a facelet tag library descriptor file is encountered that contains this element, the runtime must examine the `<namespace>` element in that same tag library descriptor and make it available for use in an XML namespace declaration in facelet pages.[P1_end_composite_library_decl]

10.3.3.2 Creating an instance of a *top level component*

[P1_start_top_level_component_creation]If, during the process of building the view, the facelet runtime encounters an element in the page using the prefix for the namespace of a composite component library, the runtime must create a `Resource` instance with a library property equal to the library name derived in Section 10.3.3.1 “Declaring a composite component library for use in a Facelet page”and call the variant of `application.createComponent()` that takes a `Resource`.

After causing the *top level component* to be instantiated, the runtime must create a `UIComponent` with component-family of `javax.faces.Panel` and renderer-type `javax.faces.Group` to be installed as a facet of the *top level component* under the facet name `UIComponent.COMPOSITE_FACET_NAME`.[P1_end_top_level_component_creation]

10.3.3.3 Populating a *top level component* instance with children

[P1_start_top_level_component_population]As specified in Section 3.6.1.3 “How does one make a composite component?” the runtime must support the use of `composite:` tag library in the *defining page* pointed to by the `Resource` derived as specified in Section 10.3.3.2 “Creating an instance of a top level component”.
[P1_start_top_level_component_population]The runtime must ensure that all `UIComponent` children in the *composite component definition* within the *defining page* are placed as children of the `UIComponent.COMPOSITE_FACET_NAME` facet of the *top level facet*.[P1_end_top_level_component_population]

Please see the tag library documentation for the `<composite:insertChildren>` and `<composite:insertFacet>` tags for details on these two tags that are relevant to populating a *top level component* instance with children.

Special handling is required for attributes declared on the *composite component tag* instance in the *using page*. [P1_start_composite_component_tag_attributes]The runtime must ensure that all such attributes are copied to the attributes map of the *top level component* instance in the following manner.

Obtain a reference to the `ExpressionFactory`, for discussion called *expressionFactory*.

Let the value of the attribute in the *using page* be *value*.

If *value* is "id" or "binding" without the quotes, skip to the next attribute.

If the value of the attribute starts with "#{" (without the quotes) call `expressionFactory.createValueExpression(elContext, value, Object.class)`

If the value of the attribute does not start with "#{", call `expressionFactory.createValueExpression(value, Object.class)`

If there already is a key in the `map` for *value*, inspect the type of the value at that key. If the type is `MethodExpression` take no action. [P1_end_composite_component_tag_attributes]

For code that handles tag attributes on `UIComponent` XHTML elements special action must be taken regarding composite components. [P1_start_composite_component_method_expression]If the type of the attribute is a `MethodExpression`, the code that takes the value of the attribute and creates an actual `MethodExpression` instance around it must take the following special action. Inspect the value of the attribute. If the EL expression string starts with the `cc` implicit object, is followed by the special string "`attrs`" (without the quotes), as specified in Section 5.6.2.2 "Composite Component Attributes ELResolver", and is followed by a single remaining expression segment, let the value of that remaining expression segment be *attrName*. In this case, the runtime must guarantee that the actual `MethodExpression` instance that is created for the tag attribute have the following behavior in its `invoke()` method.

Obtain a reference to the current composite component by calling `UIComponent.getCurrentCompositeComponent()`.

Look in the attribute of the component for a key under the value *attrName*.

There must be a value and it must be of type `MethodExpression`. If either of these conditions are `false` allow the ensuing exception to be thrown.

Call `invoke()` on the discovered `MethodExpression`, passing the arguments passed to our `invoke()` method.[P1_end_composite_component_method_expression]

[P1_start_composite_component_retargeting]Once the composite component has been populated with children, the runtime must ensure that `ViewHandler.retargetAttachedObjects()` and then `ViewHandler.retargetMethodExpressions()` is called, passing the *top level component*.[P1_end_composite_component_retargeting] The actions taken in these methods set the stage for the tag attribute behavior and the special `MethodExpression` handling behavior described previously.

[P1_start_nested_composite_components]The runtime must support the inclusion of composite components within the *composite component definition*. [P1_end_nested_composite_components].

10.4 Standard Facelet Tag Libraries

This section specifies the tag libraries that must be provided by an implementation.

10.4.1 JSF Core Tag Library

This tag library must be equivalent to the one specified in *Section 9.4 "JSF Core Tag Library"*.

For all of the tags that correspond to attached objects, the Facelets implementation supportes an additional attribute, `for`, which is intended for use when the attached object tag exists within a composite component. If present, this attribute refers to the value of one of the exposed attached objects within the composite component inside of which this tag is nested.

The following additional tags apply to the Facelet Core Tag Library only.

10.4.1.1 <f:ajax>

This tag serves two roles depending on its placement. If this tag is nested within a single component, it will associate an Ajax action with that component. If this tag is placed around a group of components it will associate an Ajax action with all components that support the “events” attribute. In there is an outer

Syntax

<f:ajax [event=”Literal”] [execute=”Literal | Value Expression”] [render=”Literal | Value Expression”] [onevent=”Literal | Value Expression”] [onerror=”Literal | Value Expression”] | [listener=”Method Expression”] [disabled=”Literal|Value Expression”] [immediate=”Literal|ValueExpression]/>

Body Content

empty.

Attributes

The following optional attributes are available:

TABLE 10-3

Name	Expr	Type	Description
event	String	String	A String identifying the type of event the Ajax action will apply to. If specified, it must be one of the events supported by the component the Ajax behavior is being applied to. If not specified, the default event is determined for the component. The default event is “action” for ActionSource components and “valueChange” for EditableValueHolder components.
execute	VE	Collection<String>	If a literal is specified, it must be a space delimited String of component identifiers and/or one of the keywords outlined in Section 14.2.2 “Keywords”. If not specified, then @this is the default. If a ValueExpression is specified, it must refer to a property that returns a Collection of Strings. Each String in the Collection must not contain spaces.
render	VE	Collection<String>	If a literal is specified, it must be a space delimited String of component identifiers and/or one of the keywords outlined in Section 14.2.2 “Keywords”. If not specified, then @none is the default . If a ValueExpression is specified, it must refer to a property that returns a Collection of Strings. Each String in the Collection must not contain spaces.

TABLE 10-3

Name	Expr	Type	Description
onevent	VE	String	The name of a JavaScript function that will handle events
onerror	VE	String	The name of a JavaScript function that will handle errors.
disabled	VE	boolean	“false” indicates the Ajax behavior script should be rendered; “true” indicates the Ajax behavior script should not be rendered. “false” is the default.
listener	ME	MethodExpression	The listener method to execute when Ajax requests are processed on he server.
immediate	VE	boolean	If “true” behavior events generated from this behavior are broadcast during Apply Request Values phase. Otherwise, the events will be broadcast during Invoke Aplications phase.

Specifying “execute”/”render” Identifiers

The String value for identifiers specified for `execute` and `render` may be specified as a search expression as outlined in the JavaDocs for UIComponent.findComponent. [P1_start_execrenderIds]The implementation must resolve these identifiers as specified for UIComponent.findComponent.[P1_end]

Constraints

This tag may be nested within any of the standard HTML components. It may also be nested within any custom component that implements the ClientBehaviorHolder interface. Refer to Section 3.7 “Component Behavior Model” for more information about this interface. [P1_start_ajaxtag_events]A TagAttibuteException must be thrown if an “event” attribute value is specified that does not match the events supported by the component type. [P1_end_ajaxtag_events] For example:

```
<h:commandButton ..>
  <f:ajax event=”valueChange”/>
</h:commandButton id=”button1” ...>
```

An attempt is made to apply a “valueChange” Ajax event to an “action” component. This is invalid and the Ajax behavior will not be applied. [P1_start_bevent]The event attribute that is specified, must be one of the events returned from the ClientBehaviorHolder component implementation of ClientBehaviorHolder.getEventNames. If an event is not specified the value returned from the component implementation of ClientBehaviorHolder.getDefaultEventName must be used. If the event is still not determined, a TagAttributeException must be thrown.[P1_end]

This tag may also serve to “ajaxify” regions of a page by nesting a group of components within it:

```
<f:ajax>
  <h:panelGrid>
    <h:inputText id=”text1”/>
    <h:commandButton id=”button1”/>
  </h:panelGrid>
</f:ajax>
```

From this example, “text1” and “button1” will have ajax behavior applied to them. The default events for these components would cause Ajax requests to fire. For “text1” a “valueChange” event would apply and for “button1” an “action” event would apply. <h:panelGrid> has no default event so in this case a behavior would not be applied.

```
<f:ajax event="click">
  <h:panelGrid id="grid1">
    <h:inputText id="text1"/>
    <h:commandButton id="button1">
      <f:ajax event="mouseover"/>
    </h:commandButton>
  </h:panelGrid>
</f:ajax>
```

From this example, “grid1” and “text1” would have ajax behavior applied for an “onclick” event. “button1” would have ajax behavior applied for both “mouseover” and “onclick” events. The “oncick” event is a supported event type for PanelGrid components.

```
<f:ajax>
    <h:commandButton id="button1">
      <f:ajax/>
    </h:commandButton>
</f:ajax>
```

For this example, the inner <f:ajax/> would apply to “button1”. The outer (wrapping) <f:ajax> would not be applied, since it is the same type of submitting behavior (AjaxBehavior) and the same event type (action).

```
<f:ajax event="click">
    <h:inputText id="text1">
      <f:ajax event="click"/>
    </h:inputText>
</f:ajax>
```

For this example, since the event types are the same, the inner <f:ajax> event overrides the outer one.

```
<f:ajax event="action">
    <h:commandButton id="button1">
      <b:greet event="action"/>
    </h:commandButton>
</f:ajax>
```

Here, there is a custom behavior “greet” attached to “button1”. the outer <f:ajax> Ajax behavior will also get applied to “button1”. But it will be applied *after* the “greet” behavior.

Description

Enable one or more components in the view to perform Ajax operations. This tag handler must create an instance of javax.faces.component.behavior.AjaxBehavior instance using the tag attribute values. If this tag is nested within a single ClientBehaviorHolder component:

If the event attribute is not specified, determine the event by calling the component’s getDefaultEventName method. If that returns null, throw an exception.

If the event attribute is specified, ensure that it is a valid event - that is one of the events contained in the Collection returned from getEventNames method. If it does not exist in this Collection, throw an exception.

Add the AjaxBehavior to the component by calling the addBehavior method, passing the event and AjaxBehavior instance.

If this tag is wrapped around component children add the AjaxBehavior instance to the data structure holding the behaviors for that component. As subsequent child components that implement the BehaviorHolder interface are evaluated, this AjaxBehavior instance must be added as a Behavior to the component. Please refer to the Javadocs for the core tag handler AjaxHandler for additional requirements.

Examples

Apply Ajax to “button1” and “text1”:

```
<f:ajax>
  <h:form>
    <h:commandButton id=”button1” ...>
    <h:inputText id=”text1” ..>
  </h:form>
</f:ajax>
```

Apply Ajax to “text1”:

```
<f:ajax event=”valueChange”>
  <h:form>
    <h:commandButton id=”button1” ...>
    <h:inputText id=”text1” ..>
  </h:form>
</f:ajax>
```

Apply Ajax to “button1”:

```
<f:ajax event=”action”>
  <h:form>
    <h:commandButton id=”button1” ...>
    <h:inputText id=”text1” ..>
  </h:form>
</f:ajax>
```

Override default Ajax action. “button1” is associated with the Ajax “execute=’cancel’” action:

```
<f:ajax event=”action” execute=”reset”>
  <h:form>
    <h:commandButton id=”button1” ...>
      <f:ajax execute=”cancel”/>
    </h:commandButton>
    <h:inputText id=”text1” ..>
  </h:form>
</f:ajax>
```

10.4.1.2 <f:event>

Allow JSF page authors to install `ComponentSystemEventListener` instances on a component in a page. Because this tag is closely tied to the event system, please see section Section 3.4.3.4 “Declarative Listener Registration” for the normative specification.

10.4.1.3 <f:metadata>

Register a facet on the parent component, which must be the `UIViewRoot`. This must be a child of the `<f:view>`. This tag must reside within the top level XHTML file for the given viewId, not in a template. The implementation must ensure that the direct child of the facet is a `UIPanel`, even if there is only one child of the facet. The implementation must set the id of the `UIPanel` to be the value of the `UIViewRoot.METADATA_FACET_NAME` symbolic constant.

10.4.1.4 <f:validateBean>

Register a BeanValidator instance on the parent EditableValueHolder UIComponent or the EditableValueHolder UIComponent whose client id matches the value of the "for" attribute when used within a composite component. If neither criteria is satisfied, save the validation groups in an attribute on the parent UIComponent to be used as defaults inherited by any BeanValidator in that branch of the component tree. Don't save the validation groups string if it is null or empty string. If the validationGroups attribute is not defined on this tag when used in an EditableValueHolder, or the value of the attribute is empty string, attempt to inherit the validation groups from the nearest parent component on which a set of validation groups is stored. If no validation groups are inherited, assume the Default validation group, javax.validation.groups.Default. If the BeanValidator is one of the default validators, then this tag simply specializes the validator by providing the list of validation groups to be used. There are two usage patterns for this tag, both shown below. The tags surrounding and nested within the `<f:validateBean>` tag, as well as the attributes of the tag itself, are show for illustrative purposes only.

Syntax

```
<h:inputText value="#{model.property}">
  <f:validateBean validationGroups=
  "javax.validation.groups.Default,app.validation.groups.Order"/>
</h:inputText>
```

or

```
<h:form>
  <f:validateBean>
    <h:inputText value="#{model.property}" />
    <h:selectOneRadio value="#{model.radioProperty}" > ... </h:selectOneRadio>
    <!-- other input components here -->
  </f:validateBean>
</h:form>
```

Body Content

Empty in the case when the Bean Validator is to be registered on a parent component.

Filled with input components when the Bean Validator is to be set on all of the ensclosing input components.

Attributes

Name	Exp	Type	Description
binding	VE	ValueExpression	A ValueExpression that evaluates to an object that implements javax.faces.validate.BeanValidator
disabled	VE	Boolean	A flag which indicates whether this validator, or a default validator with the id "javax.faces.Bean", should be permitted to be added to this component
validationGroups	VE	String	A comma-delimited of type-safe validation groups that are passed to the Bean Validation API when validating the value

Constraints

Must be nested in an EditableValueHolder or nested in a composite component and have a for attribute. Otherwise, it simply defines enables or disables the validator as a default for the branch of the component tree under the parent component and/or sets the validation group defaults for the branch. No exception is thrown if one of the first two conditions are not met, unlike other standard validators.

JSR 303 allows the user to validate a graph of objects. This version of the JSF specification does not support graph validation.

Description

Must use or extend the `javax.faces.view.facelets.ValidatorHandler` class

If not within an EditableValueHolder or composite component, store the validation groups as defaults for the current branch of the component tree, but only if the value is a non-empty string.

If the disabled attribute is true, the validator should not be added. In addition, the validatorId, if present, should be added to an exclusion list on the parent component to prevent a default validator with the same id from being registered on the component.

The createValidator() method must:

If binding is non-null, create a ValueExpression by invoking Application.createValueExpression() with binding as the expression argument, and Validator.class as the expectedType argument. Use the ValueExpression to obtain a reference to the Validator instance. If there is no exception thrown, and ValueExpression.getValue() returned a non-null object that implements javax.faces.validator.Validator, it must then cast the returned instance to javax.faces.validator.BeanValidator, configure its properties based on the specified attributes, and return the configured instance. If there was an exception thrown, rethrow the exception as a TagException.

Use the validatorId if the validator instance could not be created from the binding attribute. Call the createValidator() method of the Application instance for this application, passing validator id "javax.faces.Bean". If the binding attribute was also set, evaluate the expression into a ValueExpression and store the validator instance by calling setValue() on the ValueExpression. It must then cast the returned instance to javax.faces.validator.BeanValidator, configure its properties based on the specified attributes, and return the configured instance. If there was an exception thrown, rethrow the exception as a TagException.

10.4.1.5 <f:validateRequired>

Register a RequiredValidator instance on the parent EditableValueHolder UIComponent or the EditableValueHolder UIComponent whose client id matches the value of the "for" attribute when used within a composite component.

Syntax

<f:validateRequired/>

Body Content

empty

Attributes

Name	Exp	Type	Description
binding	VE	ValueExpression	A ValueExpression that evaluates to an object that implements javax.faces.validate.RequiredValidator
disabled	VE	Boolean	A flag which indicates whether this validator, or a default validator with the id "javax.faces.Required", should be permitted to be added to this component

Constraints

Must be nested in an EditableValueHolder or nested in a composite component and have a for attribute (Facelets only). Otherwise, it simply enables or disables the use of the validator as a default for the branch of the component tree under the parent. No exception is thrown if one of the first two conditions are not met, unlike other standard validators.

Description

Must use or extend the `javax.faces.view.facelets.ValidatorHandler` class

If the disabled attribute is true, the validator should not be added. In addition, the validatorId, if present, should be added to an exclusion list on the parent component to prevent a default validator with the same id from being registered on the component

The createValidator() method must:

If binding is non-null, create a ValueExpression by invoking Application.createValueExpression() with binding as the expression argument, and Validator.class as the expectedType argument. Use the ValueExpression to obtain a reference to the Validator instance. If there is no exception thrown, and ValueExpression.getValue() returned a non-null object that implements javax.faces.validator.Validator, it must then cast the returned instance to javax.faces.validator.RequiredValidator, configure its properties based on the specified attributes, and return the configured instance. If there was an exception thrown, rethrow the exception as a TagException..

Use the validatorId if the validator instance could not be created from the binding attribute. Call the createValidator() method of the Application instance for this application, passing validator id "javax.faces.Required". If the binding attribute was also set, evaluate the expression into a ValueExpression and store the validator instance by calling setValue() on the ValueExpression. It must then cast the returned instance to javax.faces.validator.RequiredValidator, configure its properties based on the specified attributes, and return the configured instance. If there was an exception thrown, rethrow the exception as a TagException.

10.4.2 Standard HTML RenderKit Tag Library

This tag library must be equivalent to the one specified in Section 9.5 “Standard HTML RenderKit Tag Library”.

10.4.3 Facelet Templating Tag Library

This tag library is the specified version of the ui: tag library found in pre JSF 2.0 Facelets. The specification for this library can be found in the VDLDocs for the `ui:` library.

10.4.4 Composite Component Tag Library

This tag library is used to declare composite components. The specification for this tag library can be found in the VDLDocs for the `composite:` library.

10.4.5 JSTL Core and Function Tag Libraries

Facelets exposes a subset of the JSTL Core tag library and the entirety of the JSTL Function tag library. Please see the VDLDocs for the JSTL Core and JSTL Functions tag libraries for the normative specification.

10.5 Assertions relating to the construction of the view hierarchy

[P1-start processListenerForAnnotation] When the VDL calls for the creation of a `UIComponent` instance, after calling `Application.createComponent()` to instantiate the component instance, and after calling `setRendererType()` on the newly instantiated component instance, the following action must be taken.

Obtain the `Renderer` for this component. If no `Renderer` is present, ignore the following steps.

Call `getClass()` on the `Renderer` instance and inspect if the `ListenerFor` annotation is present. If so, inspect if the `Renderer` instance implements `ComponentSystemEventListener`. If neither of these conditions are `true`, ignore the following steps.

Obtain the value of the `systemEventClass()` property of the `ListenerFor` annotation on the `Renderer` instance.

Call `subscribeToEvent()` on the `UIComponent` instance from which the `Renderer` instance was obtained, using the `systemEventClass` from the annotation as the second argument, and the `Renderer` instance as the third argument.

[P1-end]