Using Ajax with JavaServer Faces Technology

Early web applications were created mostly as static web pages.

When a static web page is updated by a client, the entire page has to reload to reflect the update.

In effect, every update needs a page reload to reflect the change.

Repetitive reloads can result in excessive network access and can impact application performance.

Technologies such as Ajax were created to overcome these deficiencies.

Ajax is an acronym for Asynchronous JavaScript and XML, a group of web technologies that enables creation of dynamic and highly responsive web applications.

Using Ajax, web applications can retrieve content from the server without interfering with the display on the client.

JavaServer Faces 2.x provides built—in support for Ajax.

This chapter describes using Ajax functionality in JavaServer Faces based web applications.

The following topics are addressed here:

Overview

Ajax refers to JavaScript and XML, technologies that are widely used for creating dynamic and asynchronous web content.

While Ajax is not limited to JavaScript and XML technologies, more often than not they are used together by web applications.

The focus of this tutorial is on JavaScript based Ajax functionality for JavaServer Faces web applications.

About Ajax

JavaScript is a dynamic scripting language for web applications.

It is an object-oriented language that allows users to add enhanced functionality to user interfaces and allows web pages to interact with clients asynchronously.

JavaScript mainly runs on the client side (such as a browser) and thereby reduces server access by clients.

When a JavaScript function sends an asynchronous request from the client to the server, the server sends back a response which is used to update the page's Document Object Model (DOM).

This response is often in the format of a XML document.

The term Ajax refers to this interaction between the client and server.

The server response need not be in XML only but can also be in other formats such as JSON.

This tutorial does not focus on the response formats.

Ajax enables asynchronous and partial updating of web applications.

Such functionality allows for highly responsive web pages that are rendered in near real time.

Ajax based web applications can access server and process information as well as retrieve data without interfering with the display and rendering of the current web page on a client (such as a browser).

Some of the advantages of using Ajax are as follows:

- Form data validation in real time, eliminating the need to submit the form for verification

- Enhanced functionality for web pages, such as username and password prompts

- Partial update of the web content, avoiding complete page reloads

Using Ajax Functionality with JavaServer Faces Technology

Ajax functionality can be added to a JavaServer Faces application in one of the following ways:

- **Adding the required JavaScript code to an application**
- **Using the built-in Ajax resource library in JavaServer Faces 2.x**

Prior to JavaServer Faces 2.x, JavaServer Faces applications provided Ajax functionality by adding the necessary JavaScript to the web page.

From JavaServer Faces 2.x, standard Ajax support is provided by a built-in JavaScript resource library.

With the support of this JavaScript resource library, JavaServer Faces standard UI components, such as buttons, labels, or text fields, can be enabled for Ajax functionality.

You can also load this resource library and use its methods directly from within the backing bean code.

The next sections of the tutorial describe the use of the built-in Ajax resource library in more detail.

In addition, because the JavaServer Faces technology component model can be extended, custom components can be created with Ajax functionality.

An Ajax version of the `guessnumber` application, `ajaxguessnumber`, is available in the example repository.

See The `ajaxguessnumber` Example Application for more information.

The Ajax specific `f:ajax` tag and its attributes are explained in the next sections.

Using Ajax with Facelets

As mentioned in the previous section, JavaServer Faces technology supports Ajax by using a built-in JavaScript resource library that is provided as part of the JavaServer Faces core libraries.

This built-in Ajax resource can be used in JavaServer Faces web applications in one of the following ways:

- By using the `f:ajax` tag along with another standard component in a Facelets application.

This method adds Ajax functionality to any UI component without additional coding and configuration.

- By using the JavaScript API method `jsf.ajax.request()`, directly within the Facelets application.

This method provides direct access to Ajax methods, and allows customized control of component behavior.

Using the `f:ajax` Tag

The `f:ajax` tag is a JavaServer Faces core tag that provides Ajax functionality to any regular UI component when used in conjunction with that component.

In the following example, Ajax behavior is added to an input component by including the `f:ajax` core tag:

```
<h:inputText
value="#{bean.message}"
>
<f:ajax event="click"/>
</h:inputText >
```

In this example, although Ajax is enabled, the other attributes of the `f:ajax` tag are not defined.

Table 12-1 lists the attributes of the `f:ajax` tag.

Table 12-1 Attributes of the `f:ajax` Tag

Name	Type	Description
`disabled`	`javax.el.ValueExpression` that evaluates to a Boolean	A Boolean value that identifies the tag status. A value of `true` indicates that the Ajax behavior should not be rendered. A value of `false` indicates that the Ajax behavior should be rendered. The default value is `false`.

`event`	`javax.el.ValueExpression` that evaluates to a `String`	A `String` that identifies the type of event to which the Ajax action will apply. If specified, it must be one of the events supported by the component. If not specified, the default event (the event that triggers the Ajax request) is determined for the component. The default event is `action` for `ActionSource` components and `valueChange` for `EditableValueHolder` components.

execute	javax.el.ValueExpression that evaluates to an Object	A Collection that identifies a list of components to be executed on the server. If a literal is specified, it must be a space-delimited String of component identifiers and/or one of the keywords. If a ValueExpression is specified, it must refer to a property that returns a Collection of String objects. If not specified, the default value is @this.

`immediate`	`javax.el.ValueExpression` that evaluates to a Boolean	A Boolean value that indicates whether inputs are to be processed early in the lifecycle. If `true`, behavior events generated from this behavior are broadcast during the Apply Request Values phase. Otherwise, the events will be broadcast during the Invoke Applications phase.
`listener`	`javax.el.MethodExpression`	The name of the listener method that is called when an `AjaxBehaviorEvent` has been broadcast for the listener.

onevent	javax.el.ValueExpression that evaluates to a String	The name of the JavaScript function that handles UI events.
onerror	javax.el.ValueExpression that evaluates to a String	The name of the JavaScript function that handles errors.
render	javax.el.ValueExpression that evaluates to an Object	A Collection that identifies a list of components to be rendered on the client. If a literal is specified, it must be a space-delimited String of component identifiers and/or one of the keywords. If a ValueExpression is specified, it must refer to a property that returns a Collection of String objects. If not specified, the default value is @none.

The keywords listed in Table 12-2 can be used with the `execute` and `render` attributes of the `f:ajax` tag.

Table 12-2 Execute and Render Keywords

Keyword	Description
`@all`	All component identifiers
`@form`	The form that encloses the component
`@none`	No component identifiers
`@this`	The element that triggered the request

Note that when you use the `f:ajax` tag in a Facelets page, the JavaScript resource library is loaded implicitly.

This resource library can also be loaded explicitly as described in Loading JavaScript as a Resource.

Sending an Ajax Request

To activate Ajax functionality, the web application must create an Ajax request and send it to the server.

The server then processes the request.

The application uses the attributes of the `f:ajax` tag listed in Table 12-1 to create the Ajax request.

The following sections explain the process of creating and sending an Ajax request using each of these attributes.

Note - Behind the scenes, the `jsf.ajax.request()` method of the JavaScript resource library collects the data provided by the Ajax tag and posts the request to the JavaServer Faces lifecycle.

Using the `event` Attribute

The `event` attribute defines the event that triggers the Ajax action.

Some of the possible values for this attribute are `click`, `keyup`, `mouseover`, `focus`, and `blur`.

If not specified, a default event based on the parent component will be applied.

The default event is `action` for `ActionSource` components such as a `commandButton`, and `valueChange` for `EditableValueHolder` components such as `inputText`.

In the following example, an Ajax tag is associated with the button component, and the event that triggers the Ajax action is a mouse click:

```
<h:commandButton id="submit"
value="Submit">
<f:ajax event="click" />
</h:commandButton>
```

```
<h:outputText id="result"
value="#{userNumberBean.response}"
/>
```

Note - You may have noticed that the listed events are very similar to JavaScript events.

In fact, they are based on JavaScript events, but do not have the `on` prefix.

For a command button, the default event is `click`, so that you do not actually need to specify `event="click"` to obtain the desired behavior.

Using the `execute` Attribute

The `execute` attribute defines the component or components to be executed on the server.

The component is identified by its `id` attribute.

You can specify more than one executable component.

If more than one component is to be executed, specify a space-delimited list of components.

The `execute` attribute can also be a keyword, such as `@all`, `@none`, `@this`, or `@form`.

The default value is `@this`, which refers to the component within which the `f:ajax` tag is nested.

The following code specifies that the `h:inputText` component with the `id` value of `userNo` should be executed when the button is clicked:

```
<h:inputText id="userNo"
value=
"#{userNumberBean.userNumber}">...
</h:inputText>
<h:commandButton id="submit"
value="Submit">
```

```
<f:ajax event="click"
execute="userNo" />
</h:commandButton>
```

Using the `immediate` Attribute

The `immediate` attribute indicates whether user inputs are to be processed early in the application lifecycle or later.

If the attribute is set to `true`, events generated from this component are broadcast during the Apply Request Values phase.

Otherwise, the events will be broadcast during the Invoke Applications phase.

If not defined, the default value of this attribute is `false`.

Using the `listener` Attribute

The `listener` attribute refers to a method expression that is executed on the server side in response to an Ajax action on the client.

The listener's `processAjaxBehavior` method is called once during the Invoke Application phase of the lifecycle.

In the following example, a `listener` attribute is defined by an `f:ajax` tag, which refers to a method from the bean.

```
<f:ajax
listener="#{mybean.someaction}"
render="somecomponent"
/>
```

The following code represents the `someaction` method in `mybean`.

```
public void
someaction(AjaxBehaviorEvent event)
{ dosomething; }
```

Monitoring Events on the Client

The ongoing Ajax requests can be monitored by using the `onevent` attribute of the `f:ajax` tag.

The value of this attribute is the name of a JavaScript function.

JavaServer Faces calls the `onevent` function at each stage of the processing of an Ajax request: begin, complete and success.

When calling the JavaScript function assigned to the `onevent` property, JavaServer Faces passes a **data** object to it.

The **data** object contains the properties listed in Table 12-3.

Table 12-3 Properties of the `onEvent` Data Object

Property	Description
`responseXML`	The response to the Ajax call in XML format
`responseText`	The response to the Ajax call in text format
`responseCode`	The response to the Ajax call in numeric code
`source`	The source of the current Ajax event: the DOM element
`status`	The status of the current Ajax call: `begin`, `success`, or `complete`
`type`	The type of the Ajax call: `event`

By using the `status` property of the data object, you can identify the current status of the Ajax request and monitor its progress.

In the following example, `monitormyajaxevent` is a JavaScript function that monitors the Ajax request sent by the event:

```
<f:ajax event="click"
render="errormessage"
onevent="monitormyajaxevent"/>
```

Handling Errors

JavaServer Faces handles Ajax errors through use of the `onerror` attribute of the `f:ajax` tag.

The value of this attribute is the name of a JavaScript function.

When there is an error in processing a Ajax request, JavaServer Faces calls the defined `onerror` JavaScript function and passes a data object to it.

The data object contains all the properties available for the `onevent` attribute, and in addition, the following properties:

- `description`
- `errorName`
- `errorMessage`

The `type` is `error`.

The `status` property of the data object contains one of the valid error values listed in Table 12-4.

Table 12-4 Valid Error Values for the Data Object `status` Property

Values	Description
`emptyResponse`	No Ajax response from server.
`httpError`	One of the valid HTTP errors: `request.status==null` or `request.status==undefined` or `request.status < 200` or `request.status >= 300`
`malformedXML`	The Ajax response is not well formed.
`serverError`	The Ajax response contains an `error` element.

In the following example, any errors that occurred in processing the Ajax request are handled by the `handlemyajaxerror` JavaScript function:

```
<f:ajax event="click" render="test"
onerror="handlemyajaxerror"/>
```

Receiving an Ajax Response

After the application sends an Ajax request, it is processed on the server side, and a response is sent back to the client.

As described earlier, Ajax allows for partial updating of web pages.

To enable such partial updating, JavaServer Faces technology allows for partial processing of the view.

The handling of the response is defined by the `render` attribute of the `f:ajax` tag.

Similar to the `execute` attribute, the `render` attribute defines which sections of the page will be updated.

The value of a `render` attribute can be one or more component `id` values, one of the keywords `@this`, `@all`, `@none`, and `@form`, or an EL expression.

In the following example, the `render` attribute simply identifies an output component to be displayed when the Ajax action has successfully completed.

```
<h:commandButton id="submit"
value="Submit"
>
<f:ajax execute="userNo"
render="result"
/>
</h:commandButton>
<h:outputText id="result"
value="#{userNumberBean.response}"
/>
```

However, more often than not, the `render` attribute is likely to be associated with an `event` attribute.

In the following example, an output component is displayed when the button component is clicked.

```
<h:commandButton id="submit"
value="Submit"
>
```

```
<f:ajax event="click"
execute="userNo" render="result"
/>
</h:commandButton>
<h:outputText id="result"
value=
"#{userNumberBean.response}"
/>
```

Note - Behind the scenes, once again the `jsf.ajax.request()` method handles the response.

It registers a response handling callback when the original request is created.

When the response is sent back to the client, the callback is invoked.

This callback automatically updates the client-side DOM to reflect the rendered response.

Ajax Request Lifecycle

An Ajax request varies from other typical JavaServer Faces requests, and its processing is also handled differently by the JavaServer Faces lifecycle.

As described in Partial Processing and Partial Rendering, when an Ajax request is received, the state associated with that request is captured by the `PartialViewContext`.

This object provides access to information such as which components are targeted for processing/rendering.

The `processPartial` method of `PartialViewContext` uses this information to perform partial component tree processing/rendering.

The `execute` attribute of the `f:ajax` tag identifies which segments of the server side component tree should be processed.

Because components can be uniquely identified in the JavaServer Faces component tree, it is easy to identify and process a single component, a few components or a whole tree.

This is made possible by the `visitTree` method of the `UIComponent` class.

The identified components then run through the JavaServer Faces request lifecycle phases.

Similar to the `execute` attribute, the `render` attribute identifies which segments of the JavaServer Faces component tree need to be rendered during the render response phase.

During the render response phase, the render attribute is examined.

The identified components are found and asked to render themselves and their children.

The components are then packaged up and sent back to the client as a response.

Grouping of Components

The previous sections describe how to associate a single UI component with Ajax functionality.

You can also associate Ajax with more than one component at a time by grouping them together on a page.

The following example shows how a number of components can be grouped by using the `f:ajax` tag.

```
<f:ajax>
 <h:form>
 <h:inputText id="input1"/>
 <h:commandButton id="Submit"/>
 </h:form>
</f:ajax>
```

In the example, neither component is associated with any Ajax `event` or `render` attributes yet.

Therefore, no action will take place in case of user input.

You can associate the above components with an `event` and a `render` attribute as follows:

```
<f:ajax event="click"
render="@all">
<h:form>
<h:inputText id="input1"
value="#{user.name}"
/>
<h:commandButton id="Submit"/>
</h:form>
</f:ajax>
```

In the updated example, when the user clicks either component, the updated results will be displayed for all components.

You can further fine tune the Ajax action by adding specific events to each of the components, in which case Ajax functionality becomes cumulative.

Consider the following example:

```
<f:ajax event="click"
render="@all">
...
<h:commandButton id="Submit">
<f:ajax event="mouseover"/>
</h:commandButton>
...
</f:ajax>
```

Now the button component will fire an Ajax action in case of a `mouseover` event as well as a mouse click event.

Loading JavaScript as a Resource

The JavaScript resource file bundled with JavaServer Faces 2.x is named `jsf.js` and is available in the `javax.faces` library.

This resource library supports Ajax functionality in JavaServer Faces applications.

In order to use this resource directly with a component or a bean class, you need to explicitly load the resource library.

The resource can be loaded in one of the following ways:

- By using the resource API directly in a Facelets page
- By using the `javax.faces.application.ResourceDependency` annotation and the resource API in a bean class

Using JavaScript API in a Facelets Application

To use the bundled JavaScript resource API directly in a web application, such as a Facelets page, you need to first identify the default JavaScript resource for the page with the help of the `h:outputScript` tag.

For example, consider the following section of a Facelets page:

```
<h:form>
<h:outputScript name="jsf.js"
library="javax.faces" target="head"
/>
</h:form>
```

Specifying the target as `head` causes the script resource to be rendered within the `head` element on the HTML page.

In the next step, identify the component to which you would like to attach the Ajax functionality.

Add the Ajax functionality to the component by using the JavaScript API.

For example, consider the following:

```
<h:form>
<h:outputScript name="jsf.js"
library="javax.faces" target="head"
>
<h:inputText id="inputname"
value="#{userBean.name}"/>
<h:outputText id="outputname"
value="#{userBean.name}"/>
```

```
<h:commandButton id="submit"
value="Submit"
onclick="
jsf.ajax.request(this, event,
{execute:'inputname',
render:'outputname'
});
return false;"
/>
</h:form>
```

The `jsf.ajax.request` method takes up to three parameters that specify source, event, and options.

The source parameter identifies the DOM element that triggered the Ajax request, typically `this`.

The optional event parameter identifies the DOM event that triggered this request.

The optional options parameter contains a set of name/value pairs from Table 12-5.

Table 12-5 Possible Values for the Options Parameter

Name	Value
`execute`	A space-delimited list of client identifiers or one of the keywords listed in Table 12-2. The identifiers reference the components that will be processed during the execute phase of the lifecycle.
`render`	A space-delimited list of client identifiers or one of the keywords listed in Table 12-2. The identifiers reference the components that will be processed during the render phase of the lifecycle.
`onevent`	A `String` that is the name of the JavaScript function to call when an event occurs.
`onerror`	A `String` that is the name of the JavaScript function to call when an error occurs.
`params`	An object that may include additional parameters to include in the request.

If no identifier is specified, the default assumed keyword for the `execute` attribute is `@this`, and for the `render` attribute it is `@none`.

You can also place the JavaScript method in a file and include it as a resource.

Using the `@ResourceDependency` Annotation in a Bean Class

Use the `javax.faces.application.ResourceDependency` annotation to cause the bean class to load the default `jsf.js` library.

To load the Ajax resource from the server side, use the `jsf.ajax.request` method within the bean class.

This method is usually used when creating a custom component or a custom renderer for a component.

The following example shows how the resource is loaded in a bean class:

```
@ResourceDependency(name="jsf.js"
library="javax.faces"target="head")
```

The `ajaxguessnumber` Example Application

To demonstrate the advantages of using Ajax, revisit the `guessnumber` example from Chapter 5, Introduction to Facelets.

If you modify this example to use Ajax, the response need not be displayed in the `response.xhtml` page.

Instead, an asynchronous call is made to the bean on the server side, and the response is displayed in the originating page by executing just the input component rather than by form submission.

The `ajaxguessnumber` Source Files

The changes to the `guessnumber` application occur in two source files, as well as the addition of a JavaScript file.

The `ajaxgreeting.xhtml` Facelets Page

The Facelets page for `ajaxguessnumber`, `ajaxgreeting.xhtml`, is almost the same as the `greeting.xhtml` page for the `guessnumber` application:

```
<h:head>
<title>
Ajax Guess Number Facelets Application
</title>
</h:head>
<h:body>
<h:form id="AjaxGuess">
<h:outputScript name="ui.js"
target="head"
/>
```

```
<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" >
```

```
<!--
<f:ajax execute="userNo"
render="result errors1"
/>
-->
<f:ajax execute="userNo"
render="result errors1"
onevent="msg"
/>
</h:commandButton>
<p></p>
```

```
<h:outputText id="result"
value="#{userNumberBean.response}"
/>
<p></p>
```

```
<h:message id="errors1"
showSummary="true"
showDetail="false"
style="color: red;
font-family: 'New Century
Schoolbook', serif;
font-style: oblique;
text-decoration: overline"
for="userNo"
/></h2>
</h:form></h:body>
```

The most important change is in the `h:commandButton` tag.

The `action` attribute is removed from the tag, and `f:ajax` tag is added.

The `f:ajax` tag specifies that when the button is clicked, the `h:inputText` component with the `id` value `userNo` is executed.

The components with the `id` values `result` and `errors1` are then rendered.

If that was all you did (as in the commented-out version of the tag), you would see the output from both the `result` and `errors1` components, although only one output is valid; if a validation error occurs, the managed bean is not executed, so the `result` output is stale.

To solve this problem, the tag also calls the JavaScript function named `msg`, in the file `ui.js`, as described in the next section.

The `h:outputScript` tag at the top of the form calls in this script.

The `ui.js` JavaScript File

The `ui.js` file specified in the `h:outputScript` tag of the `ajaxgreeting.xhtml` file is located in the `web/resources` directory of the application.

The file contains just one function, `msg`:

```
var msg = function msg(data){
var resultArea =
document.getElementById
("AjaxGuess:result");
var errorArea =
document.getElementById
("AjaxGuess:errors1");
if (errorArea.innerHTML !== null &&
errorArea.innerHTML !== ""){
resultArea.innerHTML="";
} };
```

The `msg` function obtains a handle to both the `result` and `errors1` elements.

If the `errors1` element has any content, the function erases the content of the `result` element, so that the stale output does not appear in the page.

The `UserNumberBean` *Managed Bean*

A small change is also made in the `UserNumberBean` code, so that the output component does not display any message for the default (null) value of the property `response`.

Here is the modified bean code:

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

Building, Packaging, Deploying, and Running the `ajaxguessnumber` Example

You can build, package, deploy, and run the `ajaxguessnumber` application by using either NetBeans IDE or the Ant tool.

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

This procedure builds the application into the following directory:

tut-install`/examples/web/ajaxguessnumber/build/web`

The contents of this directory are deployed to the GlassFish Server.

1. From the File menu, choose Open Project.

2. In the Open Project dialog, navigate to:

   `tut-install/examples/web/`

3. Select the `ajaxguessnumber` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

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

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

1. In a terminal window, go to:

   `tut-install/examples/web/ajaxguessnumber/`

2. Type the following command:

   ```
   ant
   ```

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

3. Type the following command:

```
ant deploy
```

Typing this command deploys `ajaxguessnumber.war` to the GlassFish Server.

To Run the `ajaxguessnumber` Example

1. In a web browser, type the following URL:

   `http://localhost:8080/ajaxguessnumber`

2. Type a value in the input field and click Submit.

If the value is in the range 0 to 10, a message states whether the guess is correct or incorrect.

If the value is outside that range, or if the value is not a number, an error message appears in red.

To see what would happen if the JavaScript function were not included, remove the comment marks from the first `f:ajax` tag in `ajaxgreeting.xhtml` and place them around the second tag, as follows:

```
<f:ajax execute="userNo"
render="result errors1"
/>
```

```
<!--
<f:ajax execute="userNo"
render="result errors1"
onevent="msg"
/>
-->
```

If you then redeploy the application, you can see that stale output from valid guesses continues to appear if you type erroneous input.

Further Information about Ajax in JavaServer Faces

For more information on JavaServer Faces Technology, see

- JavaServer Faces 2.0 technology download web site:

  http://www.oracle.com/technetwork/java/javaee/download-139288.html

- **JavaServer Faces JavaScript Library APIs are available in the Mojarra 2.x package in the following location:**

```
<mojarra-2.0.2-FCS>
/docs/jsdocs/symbols/jsf.ajax.html
```