# JavaServer Faces Training

Datum: 28. Januar 2008
Autor: Ganesh Jung
Version: 0.8
http://www.j4fry.org/

# Contents

# 1 Target audience and scope

This JavaWeb training document aims at Java developers who have a basic understanding HTML and the internet. It includes explanations of the basic Java web techniques and an introduction to JSF and it will make you understand the basic principles of JSF.

# 2 Basics of Web development with Java

## 2.1 HTTP Protokoll

HTTP requests are sent over a TCP/IP Connection. On this layer DNS names and IP-addresses are resolved. Our first request is a HTTP GET, requesting index.jsp.

An exact definition of HTTP 1.1 headers can be found at:

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

The HTTP Request starts with the request method, either GET or POST. The difference is the mechanism for passing parameters. If a HTTP GET wants to pass parameters they are appended to the request URL in the form:

```
GET /test.jsp?parameterName=parameterValue HTTP/1.1
```

The length of the HTTP GET line is limited to 2Kbyte, limiting the length of the request parameters. The solution is the HTTP POST: If a HTTP POST wants to pass parameters they are appended after 2 CR/LF's after the last request header and there is no limitation of the length:

```
POST /test.jsp HTTP/1.1
... some request headers...

parameterName=parameterValue
```

The disadvantage of the HTTP POST is the impossibility to bookmark the POST parameters in the browser. Browsers bookmark only the URL including HTTP GET parameters.

The HTTP Response starts with a success code. The response header fall into 5 categories:

Informational 1xx
Successful 2xx
Redirection 3xx
Client Error 4xx
Server Error 5xx

So far I've seen these reponse headers in practice:

200 OK - The request has succeeded
301 Moved Permanently - The requested resource has been assigned a new permanent URI
401 Unauthorized - The request requires user authentication
500 Internal Server Error - The server encountered an unexpected condition which prevented it from fulfilling the request.

### 2.1.1 HTTP GET Request

```
GET /index.jsp HTTP/1.1
Accept: */*
Accept-Language: en-gb,de;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; HVB-GROUP; .NET CLR 1.1.4322)
Host: rsq31aab
Connection: Keep-Alive
```

### 2.1.2 HTTP 401 Response with WWW-Authenticate and HTML content

```
HTTP/1.1 401 Authorization Required
Date: Thu, 13 Apr 2006 07:52:08 GMT
Server: Apache/1.3.29 (Unix) mod_jk/1.2.5 PHP/4.3.4 mod_ssl/2.8.16 OpenSSL/0.9.7c mod_gzip/1.3.26.1a
WWW-Authenticate: Basic realm="RODEO NETTOOLS port"
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1

1d0
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>401 Authorization Required</TITLE>
</HEAD><BODY>
<H1>Authorization Required</H1>
This server could not verify that you are authorized to access the document requested.  Either you supplied the
wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials
required.<P>
<HR>
<ADDRESS>Apache/1.3.29 Server at rqe3srv1 Port 1117</ADDRESS>
</BODY></HTML>

0
```

### 2.1.3 HTTP GET Request with Authorization Header

```
GET /index.jsp HTTP/1.1
Accept: */*
Accept-Language: en-gb,de;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; HVB-GROUP; .NET CLR 1.1.4322)
Host: rsq31aab
Connection: Keep-Alive
Authorization: Basic FDSA3244XFadsf=
```

### 2.1.4 HTTP 200 Response with Set-Cookie Header and chunked encoding

```
HTTP/1.1 200 OK
Date: Thu, 13 Apr 2006 08:04:39 GMT
Server: Apache/1.3.29 (Unix) mod_jk/1.2.5 PHP/4.3.4 mod_ssl/2.8.16 OpenSSL/0.9.7c mod_gzip/1.3.26.1a
Set-Cookie: JSESSIONID=547CB1ABA36BBAF1054E80DA04FF1281; Path=/RODEO_NET
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html;charset=ISO-8859-1

1ff8
```

<html>
…
</html>
0

The HTML text was omitted because of its length. You can see it by looking at the source code of RODEO_NET/index.jsp.

The response is sent "chunked", that is in pieces od size "1ff8". You see the chunk-size in the line between the HTTTP headers and start of the html. Every time a chunk is completed the size of the next chunk is sent. Thus

```
<a href=Robo/ErweiterteStammdatenAuswahl.jsp style="text-
decoration:none;color: #000000" >
```

becomes:

```
<a href=Robo/ErweiterteStammdat
8
enAuswah
1ff8
l.jsp style="text-decoration:none;color: #000000" >
```

You can see that the first chunk is 1ff8 byte sized. It is follwed by a 8 byte sized chunk and again follwed by a 1ff8 byte sized chunk.

### 2.1.5 Further HTTP GET Requests for index.jsp

To assemble index.jsp 3 further http requests are submitted:

- `GET /RODEO_NET/rq_css_qsu.jsp HTTP/1.1`
- `GET /RODEO_NET/Images/telefon.gif HTTP/1.1`
- `GET /RODEO_NET/Images/RODEOLogo.jpg HTTP/1.1`

The Cookie that was set in 2.1.4 is sent to the server with every request:

Cookie: JSESSIONID=547CB1ABA36BBAF1054E80DA04FF1281

It is a sessioncookie the browser connects internally to the IP-Adress it received the cookie from and to the HTTP session. Sessioncookies are not permanently stored but reside in the browsers memory and are discarded once the browser is closed.

The same is valid for the Authorization header. It is sent with every HTTP Request to the same server. Once you close the browser the authorization information is discarded.

There is a scope difference between sessioncookies and authorization information:
Authorization information is stored for every port of every address, the same sessioncookie is used if a connection to a different port on the same machine is made.

The responses for the images will have:

```
Content-Type: image/jpeg
```

### 2.1.6 HTTP POST Request when submitting a JSF page with input fields

POST /RODEO_NET/Roco/Monitor/OrdermonitorAuswahl.faces HTTP/1.1

Accept: */*
Referer: http://localhost/RODEO_NET/Roco/Monitor/OrdermonitorAuswahl.faces
Accept-Language: en-gb,de;q=0.5
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; HVB-GROUP; .NET CLR 1.1.4322)
Host: localhost
Content-Length: 4786
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: JSESSIONID=547CB1ABA36BBAF1054E80DA04FF1281
Authorization: Basic FDSA3244XFadsf =

jsf_tree_64=H4sIAAAAAAAAAK1UzW7TQBAemQSEVFRRDlwrgSgIyU69TpoqJ1RRlCqoEgUklEO0sbeJI
9u7Xa9LeuHGnQfgCVCfgCfgxpWX4MqZ9U%2FT2B7JReJiaT7tfD8z6738De1YwisuZyYV1J0zM7w4pS6Ld
SkC36XK55H5VjJ2omTiqkSy1zSiMyYfX4MHPBQ8YpE6%2FPDw19ejPz8NaI3h7sSd%2B4EnNQ5H45GW
sHIJq5Cw1iSsm0oMRrA5ca%2Bqg4DGsYKt0YKeUyug0czS5%2F1opo9tXB8bemfwCYwx3Jmkykq3PBiv9
RxPF8xVg6VIZGH1f0xj8P1y58f9z1%2B%2BGQBLAQBGnNoAoeCptmbOz6fxRaxYGJvyzFzEp%2BbKsPl
u%2BFJKLrXR1sT3OiLvTNJvWzNt%2FgvTCQt0upxpt8rUXjE9b2Z6ofRwp4liORkpyET63coxB8G6CNZDs
L2suJcXdtWpUW%2FYr5C004QdnWWnOcswEkkxlH6DVEZrY1rkynFW1YZ7G2mpDiMDq9PIwD0M7GM
gNge7U%2FJW2ksB2jXDtxCe6pYzsLrmDOxm1aOiqi0QI8cy2lhGe79E3kPikNpvgigSbI8Ey0icdUVyk1kRbI8
Ey0j6JfJulbxVb3E6CI%2Bzi4FYRqd0V8nqyih41vyz6DctpJFXMJWWzwuwi4E9Tf%2Bkmf6Qy1DBxrH0mA
x55CsudeN2%2BkAvzfwBXj%2F%2B3mcf33CuhPgLCrLV68AGAAA%3D&jsf_state_64=H4sIAAAAAAA
AANVba2wcVxW%2BO7vrV0zsJFAoalUL5%2F2YtXedV4PUeO043mZdp96QoCDh3t293p14dmZ85469Tu
QqRQIElIcQAZWmKj8o5RFA4gcEhAQoAcQPKgLiD6ISqhCPAhUviccfOPfOnX3gfXh2HTYdyfM4c%2BfO
d88957vnnB3f%2BBMKOxTteFfyEl7Gqo6NnDqbvkQy7MTHXnrn84P2Pl1BqGghhILOEnoS8S1UOuu2YL
Mp2iaedpimq9PYzs9gK9z9i%2B%2Fduu%2BJO0GkTKE%2B3cTZKZxhJk2gXpanxM6berZoPXJSdNO%2F
0gP7QfgLMNQ99%2Fj82fHTp6DbgxmzoOaX0%2FaqzUjBVumSesleUCkxsoQSquZZAV54biZ5FudI382XQ
x986M6LAm9RwBosw0qaGayTJ%2F%2B57YnrI%2F%2F6YxCFEqgnD1AzZpYkUXfGdAxGVxnaLhQR4Y
qIpBjVjNyJJOrhlw68g4%2B7D5ovY6phg4nLovUf2BhCDCnE4KL%2BosXQrsicmTEjM6ahwbAjsxQgF9yLc
cdewXkdxiJQ7iijHKcUryY1mxWf%2BumDz%2FwQPxdEgQQK2dplIuZAWQnxfUn%2F5TnpZig0r2VH4M3
bQE2nKDWpPScVZdm8WaiWtvsXTJohiWwCWhZLMynsIG6aOsHGj4fo1Z9f%2F%2FdrCgpcROFlrDuAJV
DkM18Dh1KJqL9y1IBsAJBNmbTg4eJNH6iDLcRQlGMpqgs4Q2x1haSxZanvSEyYBcs0iMHO4Zw6NTs3c2p
ufmI6kZycT0ym1ttiirBvTb732idvfmMsyC1jpR86H5KvG2AoDFobHZPHUfcY9Y5RV6vH3Msx2Sw2Io%2F
H3duydeywbHZEXkv52GG3WVR2eqTIR3wf372FoS6GaY4w3oSZoKOQgQvCzh5wbThSqYSMN3Z1fpwxn
MmTLLeWFMOMXKCgH0J%2FNPDSneGvfuoqTFcSbZtfEdKsaMGbMjToWjhXUIRLThRd8xjgChH2hSok
A0Iy0HCmwzVkvbVslJ%2FslxaaIjqwTDMLBSvYEp9NGBmTUmguWr0ZVeqvNz47kQdzJVl5k6G%2B%2B
OwcwVnT0FelrFjdf1D0L1yJSZMGTR9ooulKLU9%2B6Tcn1342e1tBShJ1zWd0bNswrgrymOAi4I4dVVPgc
ms1zUi%2BLVrLFO02aU7FFn%2BfWlh1oRBdPc8hxjUjC3yUKFg6Km%2Bca3YMX3FsQuPgrWoBG1mgpj
UgUjAnzTRsMfRH%2BW6WoTetb0qMNe7NUlWWJRRreZJHAdWe2qhmCMub2XqwykbB0L7hK%2BZ6A
hQgKFngy4GqGcvmIlmTbx1bZ5ISXkkeEPI%2BmLf961cJkMCr1AKQO5CqbZsZDXNdPH1t%2BtUfPPPXR
8S0BRfJqiR01wYknYNJjMDGvXf6fHxoqNiE6HyZf0So92Q9ew9z8UPi5pBgqkHZJOxyyBH3EHMPkljG3M
NRwSvxdT7Cd5MV0qnaThEQIBR%2BxsWJKrM5Wn8CbeHIJHuBaIxQHfaOkQNvwMyxhWHxLs5VmFWz
KbbK7BMWd8I1lOlL6TrM6RvBLMCTqZZ2GGlGO0q19kDd3acMp3CGrPIBwUJ%2F7LA3Hh%2FDancYh
njZ1Y2D7knlzRUAbbuoQ4w6pAO4C35x902SBezorKzvQ6MdwL3oF3e%2FxH3eDZI48vvHdZ0M1fKNDgzI
9Dug3pRJWXxVzgMXfrQF2MV2ce8GRQ6C%2FyYMy2kaMnTVo1DFpcnjMtYb2QBdMjf2ro6LE7Bk5gjd
%2FspnPvuPp95%2FTOEhuoyLvZRDtHvMKaQJfd%2BNaw9u%2BcSvPuSlUS1T8KH6FEzoAgQb3LQug4l
R7BTWiqXFe6R2EpUxjWVCwSq98Ux4AlTeRJTvm79LN%2BqnK76m%2Fzu%2BzXZGM5LEyLG864SBk
Q4423f9o8bFKtTRTfK1FkOV4RZDFUU6WEweo%2B3FJRXB%2Bnqn2N88LjlLzayzyNb4E49XPTxc%2F2
HLfYi0FMCU5L0yRuXX7%2Ba7Zxl6g1gQJChScQt2AcW9hqxwfJFpkH14J568P04uOzkbcqE8I%2F9z6bUJ
T5lG1vaOnnTrrMwGIIjXDLJOUHoz8CsTb3ZPPHlPCtoyDVLV8ql3b8tFIA5tQVvEHFTlVXHTyeDv%2FtyK
7z7tTeKzHaCBv%2FnH%2BzmJN6B0lgDe1iIBdEnHl2WTUVkPicqyyaisk4x6K%2FHdTFx21ffxtEmoTQyIc
NpNU7rEna42TSXwsN8VY%2BscsSB5Bj8V2a27bPQkIfgAz2P%2Ff2MPHPNv7O%2BREf6R4x3A%2B3b
%2FeD%2FSTiTcLt6jry%2FyC5xokfwg3BfLZPzW16WLdiBvDRz3j%2F5rm50ntUjdOxtTd4iL705Etqcp4Vo

6ZpdrhGMNioTlJwm73GZQFqwMyiDK3g9JHje2kiABsU%2FCgBQuWxKNM9QNeLisdjXQ1%2BTsEqi%2F7TuBDcp1VK6fUflzRDS2wfXzlji%2FDT0NNJjdhstpg5BZ1woaO28aFalnpFnqOWk6aZ3chczTneRguwzwFb8rcvdZzGAgcinuGlZ5JbkD1PVlv8C3QPJ53qSLZqGA5Yo8GusA8Bu%2Ba24A%2FDGcyVcgDxzpRJFqpLFXdzfzahkNR4%2FK47H2vJqh3gWN6Fm2ahFe%2BoWAcDMdPa7ZnqPzabt5L3jrq77X68BtL9X6fgds%2FQ%2F%2B8f7Ew3unA3h%2F7x%2FvLz28L3c2HtrbsXioQQJqmAXNwPqFfK5GNNSgslV%2Bjv9%2Bv5nhEJ%2Bp63z3nBf6BC9Mn2ZoQETj2Ln1PKFDqXNnSnfhHE6V0U2Iig61x58xWW6Myc8sYmPt8SfXwp%2F57i9%2BaPNg%2FVkT8hzJEWMFgp2KQGnpHuFPxX%2B21lH%2BVHzWS5Y8%2FuSxTQeqDYr%2FaoPLnxDRtBKKtR3RHGjTI73voGQdMLbROl9nPPIejGiUpdeZR9qtRzTKxQ7gpa1HNEorFalNjGj2tVicD8mPEmUxfkwul2Pex4obLSa0Uj3YsC%2FykKjNmnxI3Am1ayEvtl7RHj3aAYv%2Bgu%2BKxRmyOmvobX3g0S7oz%2FsFHZ4zISB19dw9peka1jtQIFa%2BeA8UiMPVtLAHNLKdLqkTvBxiZJt%2BAV6GWN%2BRQd86ThO9ype3DF%2FZk3IyeWLsqVg20YWy%2FhjaW9%2FfV8QXUeWvPqWnN54SMUJVsO%2Bv2xoR7%2BG3VcPZCsMZT6f5L%2BnuiCh6uHb1kuGcrqXVCWxkiH4O53a6%2FyZQ9THswgdeee2FgX0x9%2FNgnOG%2FzvEX9cEVy2v2TmDfvUnoP1LuP0KXItB%2FxO0%2FUur%2FBP8itSsjLgHV7o2hWrv%2FwMe%2F%2BeHfvSA%2BOyrZWHPtHmysXf5tx4qWZXnI%2FaIjI63p%2Bq2g69PExgVGhsR%2FCNhDNrfTBpZ0uL4l5dye3I5Sop%2F1ZlUs%2FhdrLb3o6jIAAA%3D%3D&jsf_viewid=%2FRoco%2FMonitor%2FOrdermonitorAuswahl.jsp&RqAction=Ordermonitor__id44&Ordermonitor__id1=00001&Ordermonitor__id2=-1&Ordermonitor__id8=&Ordermonitor__id11=-1&Ordermonitor__id14=-1&Ordermonitor__id21=*&Ordermonitor__id22=&Ordermonitor__id26=&Ordermonitor__id30=*&Ordermonitor__id31=&Ordermonitor__id35=&Ordermonitor__id39=EUR&Ordermonitor_SUBMIT=1

## 2.2 Servlets und JSPs

JSF is based on Servlets and JSPs. Servlet and JSP are specifications that result from SUNs Java Community Process (JCP). If a new standard is to be established a Java Specification Request (JSR) is issued.

Servlet 2.4 is specified in JSR-154, JSP 2.0 is specified in JSR-152. Tomcat 5.0 is the reference implementation for Servlet 2.4/JSP2.0. Here is the mapping between Servlet/JSP and Tomcat version:

| Servlet/JSP Spec | Apache Tomcat version |
|---|---|
| 2.4/2.0 | 5.x |
| 2.3/1.2 | 4.1.x |
| 2.2/1.1 | 3.3.x |

RODEO_NET works with Tomcat 5 which implements the Servlet 2.4 spec and the JSP 2.0 spec.

Please look at http://www.javaworld.com/javaworld/jw-03-2003/jw-0328-servlet.html to get to know more about the Servlet 2.4 spec You can download the spec at http://www.jcp.org/aboutJava/communityprocess/final/jsr154/index.html.

Look at http://www.onjava.com/pub/a/onjava/2003/11/05/jsp.html to read more about the JSP spec 2.0 and at http://www.jcp.org/aboutJava/communityprocess/final/jsr152/ you find the spec itself.

### 2.2.1 Servlet processing

To process HTTP request with Java you need a server that accepts HTTP requests and passes them to a Java method.

This is the process defined in the Servlet specification. It defines that the server is called a servlet container and that the classes the Container passes HTTP requests to are called Servlets. Servlets use the Servlet API to communicate with the servlet container. You find the documentation of the Servlet API in the weapps/tomcat-docs directory of your Tomcat installation or on the web.

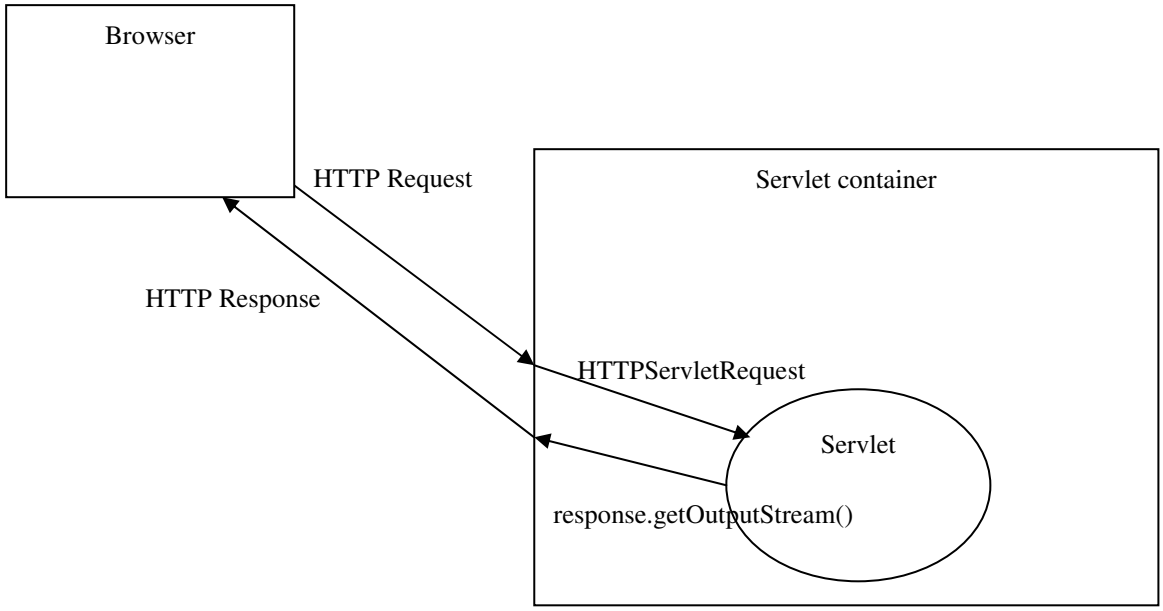
The Servlet container has a configuration file named web.xml where request patterns are mapped to Servlets. JSF is a servlet-centric framework, so the web.xml maps *.jsf (or *.faces, its just a matter of configuration) to the Faces Servlet. See 2.4.1 for details on the configuration of web.xml.

Before passing a Request to a Servlet the Servlet Container analyses the request and instantiates a Java HTTPServletRequest object that contains all the information of the request in a structured form.

In addition to the Request object the Servlet receives a HTTPServletResponse object that contains the the stream the servlet can write its response to. The Servlet can also set response headers to the HTTPServletResponse.

Internally the HTTPServletResponse will buffer the response headers and the response content until the buffer size is reached. Once the buffer size is reached the complete response header and the buffer content is sent to the browser. After the buffer has been flushed the Servlet is not allowed to change the reponse headers any more (Because the browser has already received them).

Still the HTTPServlet continues to run, thus eventually producing an Exception. In this case error handling must remain incomplete. The browser has received half an HTML page and the rest of it is not available. Sometimes the Servlet container will even send the errors stack trace to the browser. Inside the browser this will become visible when the HMTL pages source code is viewed (german IE calls this "Quelltext anzeigen", available on the right mouse button).




## 2.2.2 JSP processing

JSPs processing is based on Servlet processing. If a request asks for a .jsp the Servlet container will try to find a corresponding JSP file. The JSP will serve as a template to generate a Servlet. There are 3 basic types of code inside a JSP:

- Java code inside <% %> brackets that will be exceuted inside the generated Servlet. The browser will never see any of your Java code, it is all executed inside the Servlet container during the rendering phase of the JSP.

- JSP tags that will invoke Java classes (see 2.7). Tags are executed by the Servlet container, but may produce Javascript code that is sent to and executed by the browser. The servlet container will never look at any of your Javascript code, it is treated just like HTML code and sent to and executed by the browser.
- HTML code (including Javascript) that will be rendered to the browser with out.println("HTML code"); statements where out.println writes the HTML code to the OutputStream.

Java code inside `<% %>` brackets can have 3 forms:

`<% java statements %>` will execute the java statements.
`<%= java expression %>` will evaluate the expression and output the expressions value.
`<%! attribute definition %>` defines a class attribute for the java servlet class that is generated from the JSP.

When serving a JSP the servlet container will generate a Java class from the JSP, invoke javac to compile the Java class and pass execution control to the new Servlet. The container will need a working directory to store the generated .java and .class files.

You find the documentation of the JSP API in the weapps/tomcat-docs directory of your Tomcat installation or on the web.

## 2.3 JNDI

JNDI, the Java Naming and Directory Interface allows providing and accessing resources through a JNDI capable container. All J2EE server like BEA Weblogic, IBM Websphere and JBoss fully support JNDI.

With Tomcat you don't have full JNDI support. You cannot provide your own services through JNDI, but you can access JNDI resources provided by Tomcat.

See 2.5.2 for an example that provides a Tomcat ConnectionPool through JNDI. See 2.4.1.5 to configure a JNDI resource for use in your web application.

## 2.4 Web application configuration

The servlet specification defines a structure for web applications. The web root folder is the folder that contains the web application. It is defined in the servlet containers config file.

Inside web root and its subfolders the JSPs are located. There is one special subfolder named WEB-INF that contains all non-JSP content of the web application.

### 2.4.1 web.xml

The web.xml contains the configuration of the web application. The possible tags used to configure a servlet container are given in the order they appear in web.xml.

This is the skeleton of a web.xml to start your application with:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
      version="2.4">
</web-app>
```

#### 2.4.1.1 Context params

Context params defined in web.xml can be accessed from any Servlet through the Servlet API (see 2.2.1). using the **`getInitParameter`**`(java.lang.String name)` method. The Servlet API says: The `ServletContext` object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.

```
<context-param>
      <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
      <param-value>client</param-value>
</context-param>
```

#### 2.4.1.2 Filter and filter mapping

Filters and servlet have an order which is defined in the web.xml. Filters matching a request pattern are called consecutively in the defined order.

Filters follow the Interceptor pattern. A request that matches the pattern of a filter-mapping will trigger the doFilter () method of the filter. The doFilter () method passes the request to other servlets and filters defined in web.xml by calling doChain(). After exceution of the subsequent filters and servlets in the chain control cames back to the filter that had called doChain().

```
<filter>
      <filter-name>Hibernate Transaction Manager</filter-name>
      <filter-class>
            org.j4fry.common.servlets.TransactionManager
      </filter-class>
</filter>

<filter-mapping>
      <filter-name>Hibernate Transaction Manager</filter-name>
      <url-pattern>*.jsf</url-pattern>
</filter-mapping>
```

#### 2.4.1.3 listener

The servlet container can call listener classes at startup and shutdown of the container. Listener classes need to implement the interface ServletContextListener which contains the methods contextInitialized() and contextDestroyed().

```
<listener>
      <listener-class>
            org.apache.myfaces.webapp.StartupServletContextListener
      </listener-class>
</listener>
```

#### 2.4.1.4 servlet and servlet mapping

The URL pattern that defines which servlet class is called by which request is defined in the servlet and servlet-mapping tags:

```
<servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
```

```
</servlet>
<servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

You can define servlet init params inside the servlet tag to configiure the servlet.

#### 2.4.1.5 resources

JNDI resources defined int the web.xml can be accessed with a JNDI lookup from within the application.

```
<resource-ref>
      <description>oracle DataSource</description>
      <res-ref-name>oracleDS</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
</resource-ref>
```

#### 2.4.1.6 taglib

Tag libraries defined in the web.xml can be accessed from JSPs:

```
<taglib>
      <taglib-uri>/WEB-INF/tlds/rqjsf.tld</taglib-uri>
      <taglib-location>/WEB-INF/tlds/rqjsf.tld</taglib-location>
</taglib>
```

### 2.4.2 classes

The classes folder contains Java classes that are automatically on the applications classpath.

### 2.4.3 lib

The lib folder contains libraries that contain Java classes that are automatically on the applications classpath. Classes defined in the classes folder have precedence. If you want to replace a single class that is part of a library you can place this class in the classes folder. The servlet containers classloader will first look it up in the classes folder.

It's a matter of the standard servlet container classloader: It will start searching for a class in the classes folder and then continue to search in the libraries but it will stop searching and return the class at the first occurence it can find.

### 2.4.4 tld

The tld folder may contain tag library definitions (see 2.7).

## 2.5 Tomcat Installation and Configuration

Tomcat is an Apache project. The Apache software foundation hosts open source projects. They provide a software licence (Apache License Version 2.0) that not only allows free commercial use but also free redistribution even of non-free products. Redistribution requires that you include a copy of the Apache licence. You may include you own licence for the parts of your application that you developed yourself.

Tomcat is also SUNs reference implementation for the Servlet and JSP APIs.

### 2.5.1 Server.xml

The central configuration file of Tomcat is called server.xml and can be found in Tomcats conf folder.

### 2.5.2 Context

A Tomcat instance can host several web applications. Each application will have its own URI, eg. www.myurl.org/APP1/index.jsp and www.myurl.org/APP2/index.jsp.

A context defines a web application and the URI for its access. There are 2 different modes of context definition. Either the context is part of the server. xml or it is a defined in a separate context file. Because separate context files allow to add web applications without modification of the central server.xml the Tomcat documentation recommends separate context files.

Here is an example of a Tomcat context with URI /APP1 on a Windows machine with web root folder "webroot" and a JNDI-based Oracle connection pool:

```
<Context path="/APP1" reloadable="true" docBase="App1Folder\webroot"
workDir="App1Folder\work" >
        <Resource name="oracleDS" auth="Container"
        type="javax.sql.DataSource"
        driverClassName="oracle.jdbc.driver.OracleDriver" username="xxxx"
        password="xxxx" url="jdbc:oracle:thin:@database" maxActive="20"
        maxIdle="10" maxWait="-1" />
</Context>
```

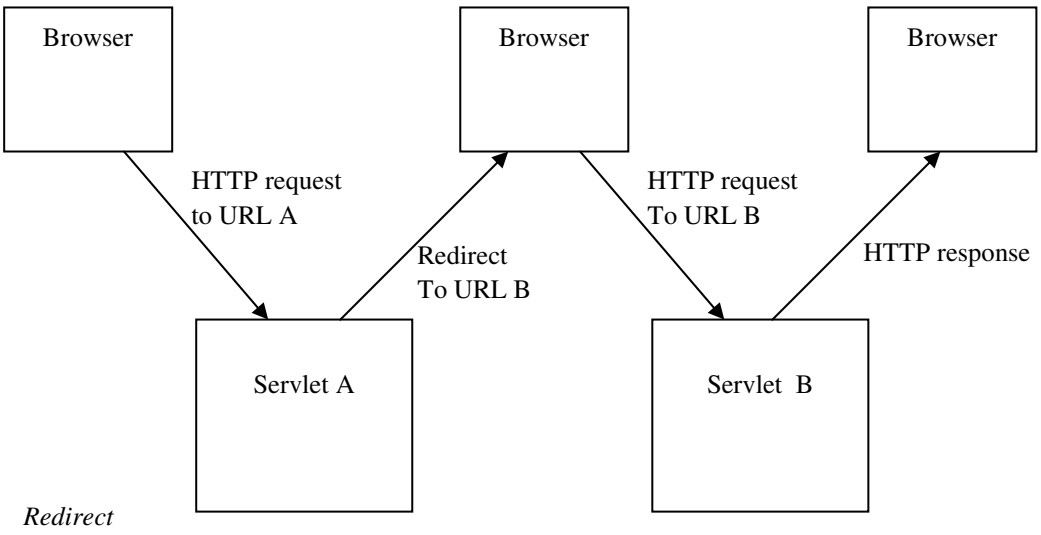
### 2.5.3 Eclipse IDE

Setting up a project has got quite a lot of pitfalls. To get a preconfigured environment for a JSF/Hibernate application including a nice bunch of examples use the example project from http://www.j4fry.org/cookbook.shtml

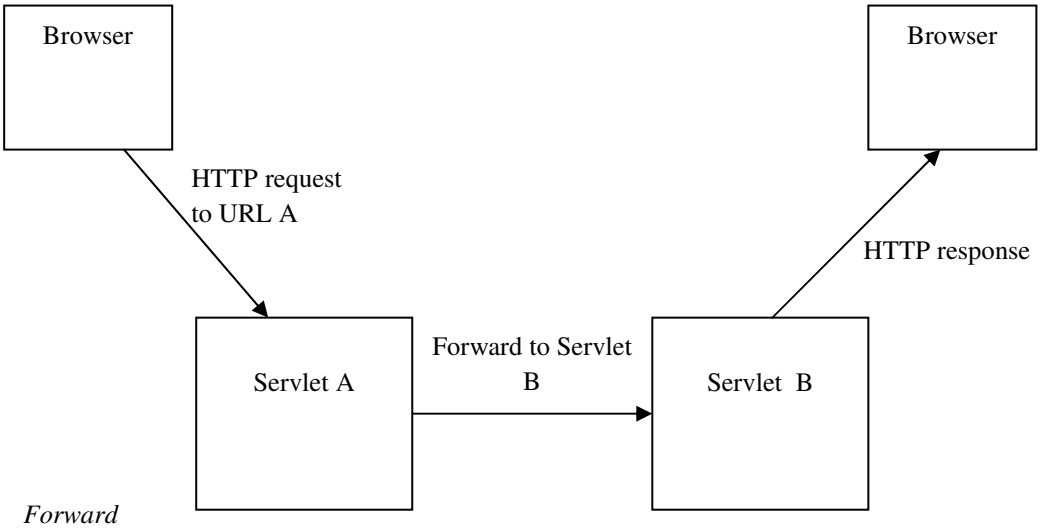
## 2.6 Redirect and forward

A Servlet or JSP that receives a request may wish to pass the execution of the request to another Servlet or JSP. There are 2 ways to do this:

- The Servlet passes execution to another Servlet without contacting the browser. This process is called a **forward**. The browser has sent the request to Servlet A and will show the URL of Servlet A, but Servlet B answers the request. Thus browser URL and HTML content will differ.
- Servlet A tells the browser to contact Servlet B to serve the request and the browser issues a new request to Servlet B. This process is called a **redirect**. Servlet A sends a HTTP 301 response along with the URL of Servlet B. The URL of Servlet B may contain request parameters, but a redirect is always a HTTP GET.




*Redirect*




*Forward*

A redirect may include the following HTTP request and responses. The request and response headers have been omitted because they don't add to the understandingof redirect and forward:

1. GET /UrlA HTTP/1.1
2. HTTP/1.1 301 Moved Permanently
   Location: /UrlB
3. GET /UrlB HTTP/1.1
4. HTTP/1.1 200 OK
   ....
   The content rendered by UrlB

The corresponding forward scenario will include only 1 request and response:

1. GET /UrlA HTTP/1.1

2. HTTP/1.1 200 OK
   ....
   The content rendered by UrlB

There are other ways to implement a redirect than the HTTP 301 redirect. You can implement a HTML redirect by using a `<meta http-equiv="refresh" content="5; URL=index.html">` tag or a Javascript redirect by using `<body onLoad="form.submit()">`. The HTML redirect can configure the number of seconds until the redirect happens (here:5). The Javascript redirect can use HTTP POST. Both may be helpful, but aren't supported by the Java API.

## 2.7 Taglib Development

A taglib is a collection of XML tags that can be used inside a JSP. Each tag is connected to a Java class that defines a `doStartTag()` and a `doEndTag()` method. When the JSP contains an XML `<tag>` the generated servlet will contain a call to the tag classes `doStartTag()` method. When the XML tag ends `</tag>` the servlet will call `doEndTag()`.

Each tag can receive parameters through its XML attributes. The attributes are passed to setter methods of the tags corresponding Java class.

The JSP needs to declare the use of of a tag library with the

```
<%@ taglib uri="taglibURI" prefix="x"%>
```

directive, where the uri designates the place where the taglib definition file can be found.

When using tags from the taglib each tag must be preceded by the prefix. If the prefix is "x" you would write `<x:myTag/>` to use tag "myTag". The choice of the prefix is technically free, but commonly used taglibs have prefixes that should be applied to them for easier recognition of the tags. See 6.2 for the standard prefixes of some JSF taglibs.

The taglibURI can be defined in the web.xml. Taglibs that are defined in the web.xml are located in a subfolder of WEB-INF. The web.xml defines the mapping between taglibURI and taglib definition file.

The taglibURI can also be defined in the taglib definition file itself. If the taglib comes with a library (which is the case for MyFaces) it is located in the libs META-INF folder. In this case it will define a taglibURI.

```
    <tlib-version>1.0.10</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>f</short-name>
    <uri>http://java.sun.com/jsf/core</uri>
    <description>
        JSF Core Tag Library
    </description>
```

The taglib definition file defines for each tag the Java class that is invoked and the possible attributes.

```
    <tag>
        <name>actionListener</name>
        <tag-class>org.apache.myfaces.taglib.core.ActionListenerTag</tag-
class>
        <body-content>empty</body-content>
        <attribute>
            <name>type</name>
            <required>true</required>
            <rtexprvalue>false</rtexprvalue>
```

```
        </attribute>
    </tag>
```

There is one attribute that is not self-explanatory: <rtexprvalue> means “runtime expression value”. If it is set to false the attribute my not contain a runtime expression value, but only a constant value. If it is set to true you could use an attribute

```
type="<%= ActionHelper.getAction(a, b, c) %>"
```

where a, b and c are variables defined somewhere in the JSP. JSF discourages the use of Java code. The standard JSF tags all have rtexprvalue>false</rtexprvalue> and the new Java EE 5 specs define possibilities to completely forbit the use of Java code in JSPs.

## 2.8 The Request object

The servlet container passes the HTTPServletRequest object to the Servlet. Through the request object the Servlet can access the request headers, the request parameters and the request attributes. Each is a map, thus containig name value pairs.

The request headers have been discussed at the beginning of this document. Names and values are of type String and are accessible through request.getHeader(name). Some special headers have their own accessor methods (getContentType, getDateHeader).

Several request parameter values can exist for one parameter name. To enable access to these values the request object has a method getParameterValues(String parameterName) that returns a String[ ]. Multiple request parameter values are seldom, thus a convenience method getParameter(String parameterName) returns only one String, which is the first element of the value array.

The third map contains the request attributes. These don’t come with the HTTP request. They can be added by the servlet itself and can be any kind of object. A servlet will store an object in the request attributes if it wants to make the objects accessibility equal the request scope, which is especially useful with forwarding (see 2.6). JavaBeans with scope=request are stored in the request attributes.

# 3 Application design

Designing an application means to define structures and resource access patterns for the code to be developed. The structures need to provide solutions for the resource access, data processing and data presentation requirements of the application.

## 3.1 Design Pattern

Writing code that is implemented along a design pattern splits complex problems into smaller parts that can be solved easier.

A design pattern isn’t a big thing. If you think up a solution for a specific problem you might come up with a similar solution. The key to Design patterns is the name a certain code structure gets. Code understanding and maintenance become easier if the classes are named along the parts that form a design pattern because everyone who knows the pattern will understand quickly how the classes are meant to work together.

The most reknown book about design patterns is “Design patterns by Erich Gamma et al.” printed by Addison Wesley often cited as [GoF] which means “Gang of Four” because there are 4 authors. The patterns in this book are not Java specific. I haven’t used all of them and some I’ve used aren’t part of [GoF]. I will shortly mention the patterns I have used in my applications, because they might be the most important ones for Java Web applications.

### 3.1.1 Abstract Factory

If you have a several parallel groups of classes and each group can provide you with a instance for a specific use the abstract factory will provide an easy way to choose the group you want the instance from. The Abstract Factory will provide a specialized Factory that produces classes of one of the possible groups. This requires Interface driven development. The runtime class will depend on the Abstract Factories configuration.

### 3.1.2 Dependency Injection

Dependency Injection (DI) wasn't known in the time of [GoF]. It is a more advanced way to define the runtime class of Interface variable and replaces the Abstract Factory pattern.

The configuration of an Abstract Factory is often done in the program code and doesn't allow fine-grained control of the factories and classes it delivers. With DI for each occurence of an Interface variable the concrete class that is used can be configured in an external resource (like a XML-file).

By using names for the objects that are injected you can share instances of injected objects among several classes. You can also define the scope of an injeted object instance and thus configure the circumstances under which a new Instance needs to be created.

DI isn't exactly a design pattern. You need a framework that reads the configuration and automatically injects the the classes into the Interface variables as needed. JSF and Spring provide a DI container.

### 3.1.3 Factory method

If you don't instantiate a class using the new operator but instead use a method that returns an instance you this a factory method. In many cases this method is named getInstance(). In a web application you may pass the HttpServletRequest object to the method and the method can decide whether to hold one instance per request, per session or per application.

If you use the Factory method pattern the constructor will often be private, thus disabling direct instantiation.

### 3.1.4 Singleton

If a class follows the Singleton pattern there is only one instance of the class in the entire application. The effect is very similar to making all methods and attributes of the class static. With the Singleton pattern you have all methods and attributes in one instance, if it's all static all methods and attributes are stored in the class object. The argument in favour of using the Singleton pattern in a Java application is the possibility to have more instances of the class without many code changes. I'm personally not so convinced that this is problem that occurs often, but many authors emphasize the advantages of a Singleton.

### 3.1.5 Builder

If many parts imformations need to be collected before a complex object can be constructed the builder pattern is a good choice. The Builder class will have setter methods to put in the parts and a getter to retrieve the ready-built object.

### 3.1.6 DAO

The DAO pattern is a Java pattern recommended by SUN. It abstracts the access to resources. You put in the identifiers and you get objects that where retrieved and instantiated by the DAO.

### 3.1.7 Adapter (or Wrapper)

If the interface of a class doesn't match the context in which you want to use it, don't worry: write an Adapter. The Adapter will pass the calls to adapted or wrapped class, often called the wrappee.

### 3.1.8 Command (or Action)

Use the Command pattern if you have a fixed place of execution where you want to pass in different similar pieces of code. Define an Command interface with an execute() method and call execute at aour fixed place of execution. Define commands that implement your interface, fill them with the parameters you need for execution and pass them on to the execution point.

### 3.1.9 MVC

MVC means Model/View/Controller and emphazises the separation of business model, user interface (view) and user triggered actions (controller).

There are lenthy discussions in the web whether MVC is a design pattern. Most authors agree that it is not, because it doesn't define the single classes that work together but rather the overall separation of application layers. In some texts the MVC pattern for web applications is called model2.

## 3.2 Java Web Frameworks

There are lots of frameworks available on the Internet. Since serveral years servlet centric frameworks are considered state of the art. Servlet centric means that all the requests are processed by one central servlet that dispatches the request.

Before JSF the dispatcher servlet forwarded the request to some controller class that initiated the application logic. There existed several approaches for automatic request parameter reading, conversion and validation.

The processing of the request parameter values has alway been the most important part, because HTTP request parameters are all Strings and can only be accessed through their request parameter names. Pre-JSF approaches used object oriented approaches that included implementation of specific classes that were automatically filled with the request values.

### 3.2.1 Struts

Struts was the most distributed web framework before JSF. It consisted of a big tag library and Java classes that dispatched the request, read the parameter values and passed control to the application. A lot of subclassing of Struts base classes was necessary to ensure collaboration of the implementation with the Struts framework.

### 3.2.2 Abaxx

Abaxx is the web framewof the internet portal of the HypoVereinbank. It is a commercial framework that was state of the art when the portal was implemented in 2000. Its base technique is simlar to Struts with less features. It also offers support for EJB 2.0 development and lookup.

### 3.2.3 Castor Pollux

Castor Pollux was the proprietary framework of the HypoVereinbank before the Unicredit merger. It is a multitier framework with 7 (!) layers.

### 3.2.4 JSF

JavaServer Faces is the most advanced Java web application framework. It combines automatic Java to HTML and HTML to Java conversion by binding JavaBeans to JSP tags. One expression and one converter combine Object reading and formatting with String parsing and Object writing. The classes that hold the converted parameter values only need to follow the JavaBean specification.

Intrusiveness is the dependency of framework based code from the framework itself. JSF is a non-intrusive framework. In a basic JSF application no JSF classes are referenced. JSF doesn't even need to be on the build classpath.

All frameworks described before are extremely intrusive because the interaction with the framework is based on inheritance and thus the applications classes need to know the frameworks class- and packagenames.

The current production version of JSF is 1.1. The MyFaces implementation is based on this version. The Java EE 5 contains JSF 1.2. With JSF 1.2 better integration with JSP and JSTL (the Java standard tag library) will be available. JSF 2.0 is currently being specified.

### 3.2.5 Spring

Spring is a framework that wants to ease development by providing services similar to J2EE but with simpler code, highly configuration based and little Intrusiveness. They provide a DI container (see 3.1.2), AOP and transaction support and a basic web framework. They also provide integration code for other frameworks (e.g. JSF and Hibernate).

As much as on one side Spring may be helpful for many purposes it also has a downside: It aims do integrate everything but it's only central concept is DI. As opposed to J2EE they don't follow a specification but put together lots of things they find useful.

### 3.2.6 Ajax

Ajax is not a framework. It is a general name for technologies. These technologies

- use taglibs
- JavaScript
- XMLHttpRequest and
- DHTML

to create web UIs with extremely flexible components that are nearly as powerfull as GUI Window components.

XMLHttpRequest is a a JavaScript technology that allows to send a XML request to the webserver and retrieve some data. With DHTML these data are used to change the page the browser has currently loaded without the need to reload the complete page. The complexity of components using XMLHttpRequest and DHTML is encapsulated in a taglib.

# 4 Basics of JSF development

JSF is the new programming model for Java web applications. Like all Java products by SUN it's a complex specification that can be implemented by different independent vendors. SUN has implemented a reference implementation (RI), the Apache foundation offers MyFaces, a JSF implementation under the Apache 2.0 licence that allows free commercial redistribution and IBM has implemented JSF and sells it with their Websphere server.

We will discuss JSF features in general and if we refer to implementation specific details we refer to Apache MyFaces.

## 4.1 Managed Beans

JSF shields the developer from the process of request parameter handling. Objects are converted, formatted and inserted in a HTML page and when the user has submitted the page the data are converted back, validated and written back to the application model.

The objects forming the application model must follow the JavaBeans specification to allow JSF automatic read and write access. JavaBeans follow only 2 rules:

- They must have a no-argument constructor
- Properties are accessed using setter and getter methods where a String property test is read with public String getTest() and written with public void setTest(String test).

To allow bean references from within a JSP the bean usage is defined in a XML file named faces-config.xml. Each bean has a name, a bean class and a bean scope.

There a 3 scopes: Request, Session and Application. If a beans has application scope JSF creates only one instance of the bean class. Any code in the application that references the beans name will reference this one instance. If the bean has session scope a seperate instance of the bean class willl be instantiated for each HTTP user session. With request scope a new instance will be associated with the request.

```
<managed-bean>
      <managed-bean-name>testBean</managed-bean-name>
      <managed-bean-class>org.test.TestBean</managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

If a bean has a propety that should automatically contain another bean the container can automatically injects the second bean when it instantiates the first bean.

```
      <managed-property>
            <property-name>dateOfArrival</property-name>
            <value>#{travelBean.firstFlight}</value>
     </managed-property>
```

## 4.2 The EL

To reference beans from a JSP the expression language (EL) is used. It allows much more that access to bean properties. If the bean property is another bean the EL allows access to the property of a property of a bean. If the bean property is a Map it allows access to the Maps elements and the key can be a property of another bean. EL expressions are marked with #{EL-expression}.

You can combine the EL constructs to reference complex model structures. One EL expression can cover two Java expressions: A call to the referenced properties getter method and a call to the setter method. Only a small subset of EL expressions is not capable of expressing a setter access.

| EL structure | Example |
|---|---|
| property access (getter and setter) | #{testBean.myProperty} |
| map access (getter and setter) | #{testBean.myMap['testKey']} |
| list access (getter and setter) | #{testBean.myList[3]} |
| String concatination | #{'test' + testBean.myString} |
| Conditional (getter only) | Bean1.myProperty == 'test' ? bean2.testProp : 'OtherResult' |

In many cases an EL expression provides a call to a getter when a page is displayed and a call to a setter when submitted values are written back to the model. Use the “getter only” types of EL epressions only in conjunction with getter only types of tag attributes. Here are some examples of getter only attributes:

- The title of a button
- The style class of a input field
- The value of a output field

These examples are all used to reference values and thus are calles value binding expressions. The El is also used to reference bean method. This is the case for buttons that should trigger actions when pushed. This is called metod binding.

The following list is copied from SUNs Java EE 5 tutorial page 126/127.

EL operators:

• Arithmetic: +, - (binary), *, / and div, % and mod, - (unary)
• Logical: and, &&, or, ||, not, !
• Relational: ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le. Comparisons can be made against other values, or against boolean, string, integer, or floating point literals.
• Empty: The empty operator is a prefix operation that can be used to determine whether a value is null or empty.
• Conditional: A ? B : C. Evaluate B or C, depending on the result of the evaluation of A.

The precedence of operators highest to lowest, left to right is as follows:

• [ ] .
• ( ) - Used to change the precedence of operators.
• - (unary) not ! empty
• * / div % mod
• + - (binary)
• < > <= >= lt gt le ge
• == != eq ne
• && and
• || or
• ? :

## 4.3 Tag – Component – Renderer

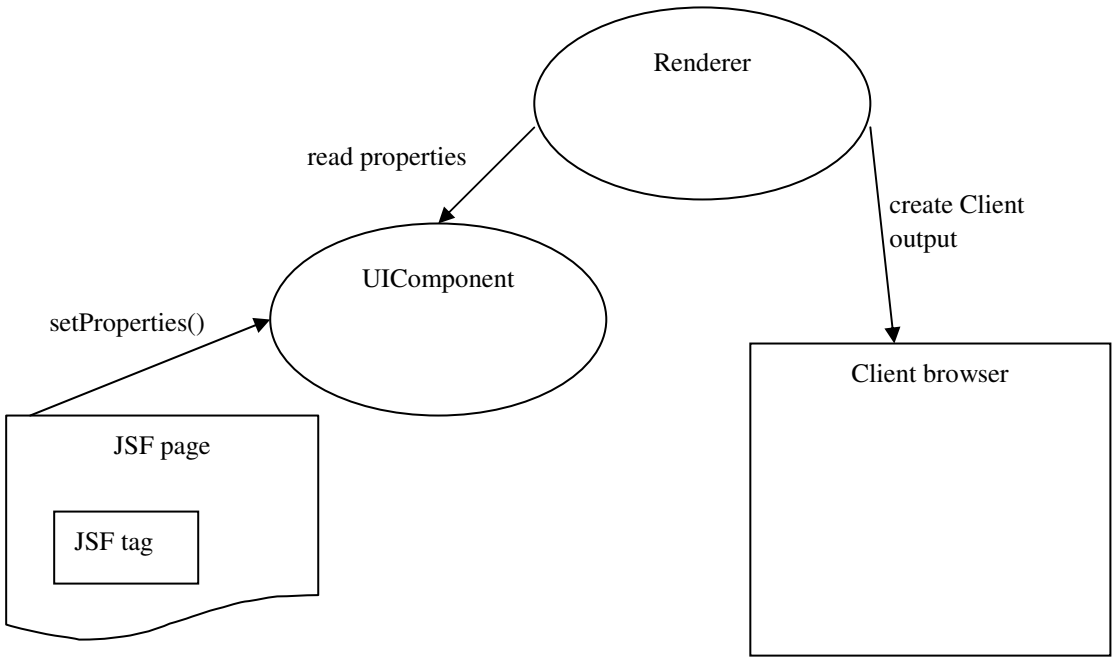
JSF Tags follow a more complex pattern than JSP tags. Each tag definition needs a corresponding UIComponent class. The tag class doesn’t have a doStartTag() method but a setProperties() method that propagates the tag attributes to the UIComponent.

JSF provides a sophisticated class hierarchy UIComponent classes are derived from.

- UIOutput is used in conjunction with tags that reference a model value that is displayed but not written back to the model.
- UIInput extends UIOutput and supports the process of writing the submitted value back to the model.
- UIData extends UIInput and supports iteration over a List of objects.
- UISelect extends UIInput and references a List of option values and a model value.

Rendering is done by a renderer class that reads the data to be rendered from the UIComponent. The renderer class has got an encodeBegin() and an encodeEnd() method that contain the rendering code. To decode the users input the renderer contains a decode() method.

If the UIComponent isn't meant to cooperate with different renderers it is possible to implement the encode and decode methods directly into the UIComponent thus surpassing the renderer layer. This proceeding is not recommended, because it limit the extensibility and mixes the layers.




## 4.4 The component tree

The JSF tags that form the page build a tree structure. Each tag has a corresponding UIComponent. The component build a tree structure. The structure of the tags is mirrored in the structure of the components.

The tags write their attributes to the components when the framework calls their setProperties() method. After rendering a page JSF serializes the component tree and either saves it to the HTTP session or sends it to the client browser in a hidden HTML input field. Saving the tree to the HTML page will reset the tree to its former state when the user clicks the browsers back button. Saving the tree to the HTTP session reduces network traffic. Network traffic increases only marginally (~50kb for an average page) compared to the size of the pictures included in many pages so I recommend saving the tree in the HTML page. 5.2 explains how to configure where the state is saved.

When the user submits a page JSF matches the HTTP parameters with the corresponding components using the component tree. The process of matching the parameters with the tree iterates over the tree and for each UIComponent it calls the corresponding renderers decode() method. In some cases it is important which JavaBean setter is called first – you can rely on the components being evaluated in the order their corresponding tags have in the JSP.

Please see 4.8 for further details on what happens to the decoded parameter value.

## 4.5 Navigation

JSF configures page navigation in the XML configuration.

If any converter or validator fails of if an action decides to set the error state JSF will automatically redisplay the same page again. This is easy because the decoding process saves the submitted parameter values in the components if the component couldn't update the model. The renderers are coded so that they check the submittedValue of the component before asking the component for the model value.

Each component that can trigger actions can have many actionListeners but only one action. This action can be either a method returning a String that is invoked through EL method binding or a simple String value. This action String decides which page will be invoked next.

Please see 5.3.2 to see how to configure page navigation in JSF.

If no navigation rules can be applied JSF will automatically roundtrip to the same page it came from. This is important to know if you want to redisplay data on the same page they where entered in: Simply let your action return a value that doesn't trigger any navigation rule and your page will be redisplayed.

## 4.6 Eventhandling

There are 2 basic kinds of events in JSF: action events and value change events. actionListeners and valueChangeListeners are defined in the

```
<f:valueChangeListener type="listeners.NameChanged" />
```

and

```
<f:actionListener type="listeners.LocaleChange" />
```

tags that are nested inside the JSF tags they should listen to. Both support two kinds of attributes. The "type=..." attribute wants a full package and class name of a class that implements ActionListener/ValueChangeListener. You will rather be using the "binding=..." attribute where you can reference a bean through the EL.

An *action event* occurs when the user activates a component that implements ActionSource. These components include buttons and hyperlinks. A *value-change* event occurs when the user changes the value of a component represented by UIInput or one of its subclasses. An example is selecting a checkbox, an action that results in the component's value changing to true. The component types that can generate these types of events are the UIInput, UISelectOne, UISelectMany, and UISelectBoolean components. Value-change events are fired only if no validation errors were detected.

## 4.7 Validators and Converters

JSF contains only 3 predefined Validator classes. They are nested inside the tag they should be used for:

```
<f:validateLongRange minimum="1"/>
```

All 3 of them support a minimum and a maximum attribute.

| Validator Class | Tag | Function |
|---|---|---|
| DoubleRangeValidator | validateDoubleRange | Checks whether the local value of a component is within a certain range. The value must be floating-point or convertible to floating-point. |
| LengthValidator | validateLength | Checks whether the length of a component's local value is within a certain range. The value must be a java.lang.String. |
| LongRangeValidator | validateLongRange | Checks whether the local value of a component is within a certain range. The value must be any numeric type or String that can be converted to a long. |

If you want to provide your own validation method you use the tags validator attribute:

```
<h:inputText id="email" value="#{checkoutFormBean.email}" size="25" maxlength="125"
validator="#{checkoutFormBean.validateEmail}"/>
```

The referenced methods interface must conform to the examples parameters:

```
public void validateEmail(FacesContext context, UIComponent toValidate,
      Object value) {
            String message = "";
            String email = (String) value;
            if (!email.contains('@')) {
                  ((UIInput) toValidate).setValid(false);
                  message = CoffeeBreakBean.loadErrorMessage(context,
                              CoffeeBreakBean.CB_RESOURCE_BUNDLE_NAME,
"EMailError");
                  context.addMessage(toValidate.getClientId(context),
                              new FacesMessage(message));
            }
      }
```

Here are the valid default JSF converters. The Java classes they convert from and to are easily recongnized by the converters names. Use the converter that fits your bean property type:

- BigDecimalConverter
- BigIntegerConverter
- BooleanConverter
- ByteConverter
- CharacterConverter
- DateTimeConverter
- DoubleConverter
- FloatConverter
- IntegerConverter
- LongConverter

• NumberConverter
• ShortConverter

Reference your converter according to this example:

```
<h:inputText id="ccno" size="19" converter=" IntegerConverter "
required="true">
...
</h:inputText>
```

See 5.3.3 for the registration of a custom converter with JSF.

## 4.8 The JSF LifeCycle

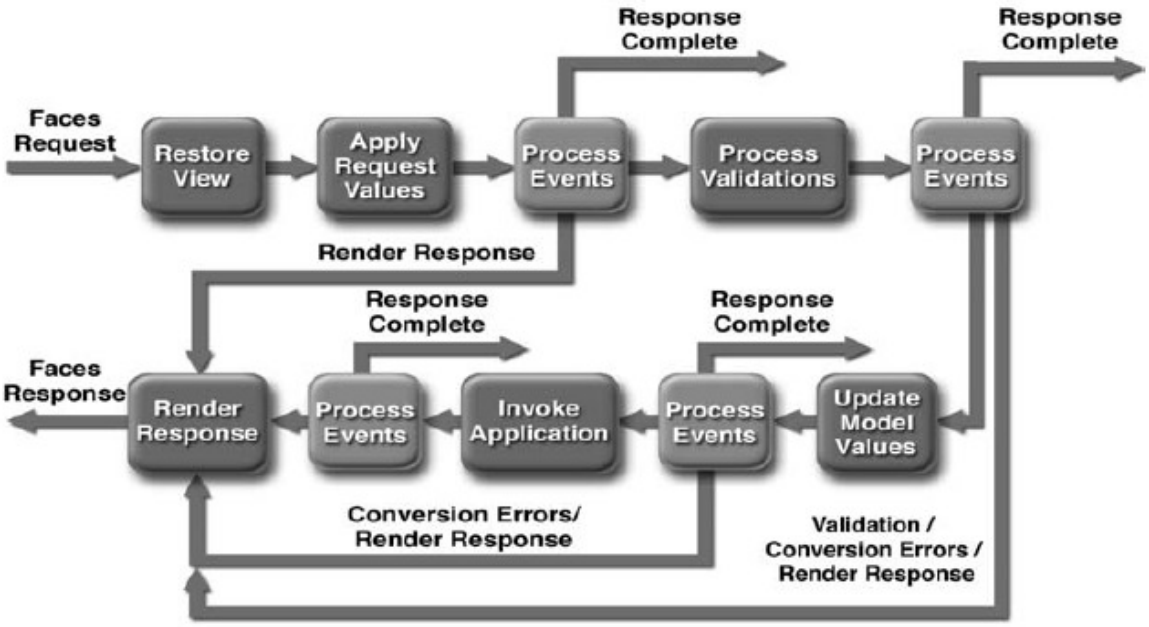
To debug a JSF application it is important to understand the JSF lifecycle. When the user has entered values in the page and submitted them a new JSF LifeCycle starts. The FacesServlet receives the request and invokes the 6 JSF LifeCycle phases. We will go into detail s for these phases to ease location of coding errors when debugging the application.

The process is described nicely in SUNs Java EE 5 tutorial, page 301-305. I've copied it into this script. The copied part goes from here to the beginning of chapter 4.9.

The parts that are written in *italics* mark an example that was added to SUNs explanations.

*The example assumes an* `inputText` *JSF tag that references a* `Double` *value and a* `commandButton` *JSF tag that references an action. It includes formatting the model value, validating the user input, converting the user input, invoking the action method and rendering the target view .*

*The example starts at the last lifecycle phase at 4.8.6, because the first time a page is rendered the phases 1-5 are skipped.*




The lifecycle handles both kinds of requests: *initial requests* and *postbacks*. When a user makes an initial request for a page, he or she is requesting the page for the first time. When a user executes a postback, he or she

submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request. When the life cycle handles an initial request, it only executes the restore view and render response phases because there is no user input or actions to process. Conversely, when the life cycle handles a postback, it executes all of the phases.

### 4.8.1 Restore View Phase

When a request for a JavaServer Faces page is made, such as when a link or a button is clicked, the JavaServer Faces implementation begins the restore view phase.

During this phase, the JavaServer Faces implementation builds the view of the page, wires event handlers and validators to components in the view, and saves the view in the FacesContext instance, which contains all the information needed to process a single request. All the application's component tags, event handlers, converters, and validators have access to the FacesContext instance.

If the request for the page is an initial request, the JavaServer Faces implementation creates an empty view during this phase and the life cycle advances to the render response phase. The empty view will be populated when the page is processed during a postback.

If the request for the page is a postback, a view corresponding to this page already exists. During this phase, the JavaServer Faces implementation restores the view by using the state information saved on the client or the server. The view for the greeting.jsp page of the guessNumber example would have the UIView component at the root of the tree, with helloForm as its child and the rest of the JavaServer Faces UI components as children of helloForm.

*When the user has submitted the example JSF page the JSF Servlet is invoked to process the request. The JSF state and the JSF component tree are restored and the tree will contain a* `UIInput` *and a* `UICommand` *component nested in a* `UIForm` *and a* `ViewRoot` *component.*

### 4.8.2 Apply Request Values Phase

After the component tree is restored, each component in the tree extracts its new value from the request parameters by using its decode method. The value is then stored locally on the component. If the conversion of the value fails, an error message associated with the component is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors resulting from the process validations phase. In the case of the userNo component on the greeting.jsp page, the value is whatever the user entered in the field. Because the object property bound to the component has an Integer type, the JavaServer Faces implementation converts the value from a String to an Integer.

If any decode methods or event listeners called renderResponse on the current FacesContext instance, the JavaServer Faces implementation skips to the render response phase.

If events have been queued during this phase, the JavaServer Faces implementation broadcasts the events to interested listeners.

If some components on the page have their immediate attributes set to true, then the validation, conversion, and events associated with these components will be processed during this phase. At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call FacesContext.responseComplete.

At the end of this phase, the components are set to their new values, and messages and events have been queued.

*When the component tree is traversed the decode methods of the inputRenderer and commandRenderer are passed the* `UIInput` *and a* `UICommand` *components of our example. The* `UIInput` *is filled the users submitted value and the action referenced by the* `UICommand` *is queued for execution.*

### 4.8.3 Process Validations Phase

During this phase, the JavaServer Faces implementation processes all validators registered on the components in the tree. It examines the component attributes that specify the rules for the validation and compares these rules to the local value stored for the component.

If the local value is invalid, the JavaServer Faces implementation adds an error message to the FacesContext instance, and the life cycle advances directly to the render response phase so that the page is rendered again with the error messages displayed. If there were conversion errors from the apply request values phase, the messages for these errors are also displayed.

If any validate methods or event listeners called renderResponse on the current FacesContext, the JavaServer Faces implementation skips to the render response phase.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call FacesContext.responseComplete.

If events have been queued during this phase, the JavaServer Faces implementation broadcasts them to interested listeners.

In the case of the greeting.jsp page, the JavaServer Faces implementation processes the standard validator registered on the userNo inputText tag. It verifies that the data the user entered in the text field is an integer in the range 0 to 10. If the data is invalid or if conversion errors occurred during the apply request values phase, processing jumps to the render response phase, during which the greeting.jsp page is rendered again, with the validation and conversion error messages displayed in the component associated with the message tag.

*If our example* `inputText` *JSF Field contains any validators these will be invoked now. If the validation fails the submitted value is redisplayed and no action is invoked .*

### 4.8.4 Update Model Values Phase

After the JavaServer Faces implementation determines that the data is valid, it can walk the component tree and set the corresponding server-side object properties to the components' local values. The JavaServer Faces implementation will update only the bean properties pointed at by an input component's value attribute. If the local data cannot be converted to the types specified by the bean properties, the life cycle advances directly to the render response phase so that the page is rerendered with errors displayed. This is similar to what happens with validation errors.

If any updateModels methods or any listeners called renderResponse on the current FacesContext instance, the JavaServer Faces implementation skips to the render response phase.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call FacesContext.responseComplete.

If events have been queued during this phase, the JavaServer Faces implementation broadcasts them to interested listeners.

At this stage, the userNo property of the UserNumberBean is set to the local value of the userNumber component.

*Now the exampleUIInputs submitted value is converted and written to the data model. Converting and settting the value to the model may also produce errors that stop the lifecycle, redisplay the submitted value and hinder execution of the action.*

### 4.8.5 Invoke Application Phase

During this phase, the JavaServer Faces implementation handles any applicationlevel events, such as submitting a form or linking to another page. At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call FacesContext.responseComplete.

If the view being processed was reconstructed from state information from a previous request and if a component has fired an event, these events are broadcast to interested listeners.

The greeting.jsp page from the guessNumber example has one applicationlevel event associated with the UICommand component. When processing this event, a default ActionListener implementation retrieves the outcome, success, from the component's action attribute. The listener passes the outcome to the default NavigationHandler. The NavigationHandler matches the outcome to the proper navigation rule defined in the application's application configuration resource file to determine which page needs to be displayed next. See Configuring Navigation Rules (4.5) for more information on managing page navigation. The JavaServer Faces implementation then sets the response view to that of the new page. Finally, the JavaServer Faces implementation transfers control to the render response phase.

*Now the examples action is invoked. The actions return value is applied to the navigation rules defined in the faces-config.xml to determine which URL is invoked next.*

### 4.8.6 Render Response Phase

During this phase, the JavaServer Faces implementation delegates authority for rendering the page to the JSP container if the application is using JSP pages. If this is an initial request, the components represented on the page will be added to the component tree as the JSP container executes the page. If this is not an initial request, the components are already added to the tree so they needn't be added again. In either case, the components will render themselves as the JSP container traverses the tags in the page.

If the request is a postback and errors were encountered during the apply request values phase, process validations phase, or update model values phase, the original page is rendered during this phase. If the pages contain message or messages tags, any queued error messages are displayed on the page.

After the content of the view is rendered, the state of the response is saved so that subsequent requests can access it and it is available to the restore view phase. In the case of the guessNumber example, if a request for the greeting.jsp page is an initial request, the view representing this page is built and saved in Faces- Context during the restore view phase and then rendered during this phase. If a request for the page is a postback (such as when the user enters some invalid data and clicks Submit), the tree is rebuilt during the restore view phase and continues through the request processing life cycle phases.

*In our example the JSF is initially invoked by a browser URL. This means that phases 2-5 are skipped and the page is directly rendered. The renderer calls the JSF container to resolve the bean reference, invokes the converter and displays the formatted value on the JSF page.*

## 4.9 Error handling

Application specific errors can occur in the phases 3-5 (4.8.3 - 4.8.5) where user input is converted and validated and the application is invoked. In the other phases errors can result from coding errors in the component and renderer implementations and from configuration errors.

Validator and Converter errors are raised by ValidatorException and ConverterException that contain a FacesMessage. JSF will catch the exception, add the FacesMessage to the FacesContext and skip to the render response phase. This means that JSF doesn't display the applications error page but induces a roundtrip that redisplays the page. Debugging tip: If your page has no means of displaying the FacesMessages that are contained in the FacesContext you may wonder at this point why your actions aren't invoked by the application.

Errors that occur during action processing shouldn't throw uncaught exceptions, because this would make JSF forward to the errors page. If your action code detects an error situation and wants to inform the user you should rather instantiate a FacesMessage and add it to the FacesContext. Then return a String that doesn't trigger any navigation rule to roundtrip to the same page and display the error message to the user.

# 5 MyFaces installation and configuration

## 5.1 libs

This is the list Java archives (jars) you need to run MyFaces 1.1.5:

From Apache Jakarta:

```
commons-beanutils-1.7.0.jar
commons-codec-1.3.jar
commons-el-1.0.jar
commons-collections-3.2.jar
commons-digester-1.8.jar
commons-fileupload-1.0.jar
commons-lang-2.1.jar
jakarta-oro.jar
```

From SUN J2EE:

```
jstl-1.1.0.jar
```

From Apache MyFaces:

```
myfaces-api-1.1.5.jar
myfaces-impl-1.1.5.jar
tomahawk-1.1.6.jar
```

## 5.2 MyFaces params in web.xml

### 5.2.1 listener

To run MyFaces you need a ContextListener for initialization:

```
<listener>
      <listener-class>
            org.apache.myfaces.webapp.StartupServletContextListener
      </listener-class>
</listener>
```

### 5.2.2 Faces Servlet

The Faces Servlet proceses all JSF requests. You are free to choose the extension for your JSF pages (usually *.jsf or *.faces):

```
<servlet>
	<servlet-name>Faces Servlet</servlet-name>
	<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
	<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

### 5.2.3 Context params

JSF can be configured via the web.xml context params. The most important setting is the State-saving-method:

```
<context-param>
	<param-name>javax.faces.STATE_SAVING_METHOD</param-name>
	<param-value>client</param-value>
</context-param>
```

You can reference multiple faces config files by using:

```
<context-param>
	<param-name>javax.faces.CONFIG_FILES</param-name>
	<param-value>robo-config.xml, roco-config.xml, common-
config.xml</param-value>
</context-param>
```

## 5.3 faces-config.xml

You can have more than one application configuration resource file. The JavaServer Faces implementation finds the file or files by looking for the following:

- A resource named `/META-INF/faces-config.xml` in any of the JAR files in the web application's `/WEB-INF/lib/` directory and in parent class loaders. If a resource with this name exists, it is loaded as a configuration resource. This method is practical for a packaged library containing some components and renderers.
- A context initialization parameter, `javax.faces.application.CONFIG_FILES`, that specifies one or more (comma-delimited) paths to multiple configuration files for your web application. This method will most likely be used for enterprise-scale applications that delegate to separate groups the responsibility for maintaining the file for each portion of a big application.
- A resource named `faces-config.xml` in the /WEB-INF/ directory of your application. This is the way most simple applications will make their configuration files available.

Here is the skeleton of a faces-config.xml to start your application with:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd" >
<faces-config>
```

```
</faces-config>
```

### 5.3.1 managed beans

The central feature of JSF is the capability to define JavaBeans in a central repository. These beans can be referenced through the EL and can be configured using dependency injection.

```
<managed-bean>
	<managed-bean-name>testBean</managed-bean-name>
	<managed-bean-class>
		org.j4fry.TestBean
	</managed-bean-class>
	<managed-bean-scope>request</managed-bean-scope>
	<managed-property>
		<property-name>testProperty</property-name>
		<value>#{secondBean}</value>
	</managed-property>
</managed-bean>
```

### 5.3.2 navigation rules

You can use a wildcard * for the from-view-id. There can several navigation-cases for one from-view-id which are the several navigation routes depending on the from-outcome.

```
<navigation-rule>
	<from-view-id>/greeting.jsp</from-view-id>
	<navigation-case>
		<from-outcome>success</from-outcome>
		<to-view-id>/response.jsp</to-view-id>
	</navigation-case>
</navigation-rule>
```

### 5.3.3 converter

You may define custom Converters and Validators in a similar way. I only provide a converter example because custom Validators are easier to provide throgh bean methods (see 4.7):

```
<converter>
	<description>
		Converter for credit card numbers
	</description>
	<converter-id>CreditCardConverter</converter-id>
	<converter-class>
		converters.CreditCardConverter
	</converter-class>
</converter>
```

### 5.3.4 components

If you write custom JSF tags you must also provide custom components:

```
<component>
	<component-type>DemoArea</component-type>
	<component-class>components.AreaComponent</component-class>
</component>
```

### 5.3.5 rendererkits

If you omit the renderer-kit id JSF assumes the defined renderer belongs to the default HTML renderer-kit. The renderKit cannot be changed when using JSF 1.1 spec conformant implementations. Additional renderkit will become available with JSF 1.2.

```
<render-kit id="PDF">
      <renderer>
            <component-family>Area</component-family>
            <renderer-type>DemoArea</renderer-type>
            <renderer-class>renderers.AreaRenderer</renderer-class>
      </renderer>
</render-kit>
```

# 6 JSF Components

## 6.1 Why develop new components?

The header is somewhat misleading: This is about developing new tags, UIComponents and renderers. Each of these can be exchanged seperately. Through the faces-config.xml you can configure which tag uses which component and which renderer.

- Exchanging the tag is necessary if the set of attributes the JSP tag supports isn't appropriate for your requirements. This can be done by extending an existing JSF tag or by implementing a new one.
- Exchanging (usually extending) a UIComponent becomes necessary if you need to change the way the component interacts with the JSF framework which is done through interfaces the UIComponent extends.
- Exchanging the renderer becomes necessary if the renderer isn't able to produce the desired output. Extending an existing renderer normally isn't possible because you need to insert or exchange rendering code somewhere in the middle of the encodeBegin() or encodeEnd() methods. Only if you want to add code before or after a tag reuse of an existing renderer may be possible.

### 6.1.1 Actions for Inputfields

In some applications we need inputfiels that trigger a JSF action when their value changes.

- The tag needs to support an action attribute to reference the JSF action that is to be triggered.
- The basic UIComponent for input fields is UIInput. It is extended to implement ActionSource to enable the component to queue the action defined in the tag.
- The HTML code needs to contain code, that reacts on the onChange event, sets an action parameter that indicates which inputfield was updated and submits the form. This requires a specific encoding and a decoding that queues the event and thus requires implementation of a renderer.

### 6.1.2 Lookup services

Another application may provide a key and the tag is required to show the associated value. This doesn't impose a change to the existing JSF components. You could instead implement a LookupBean that is referenced in the tag and receives the key that is looked up through an EL expression.

This EL expression can become complicated if the lookup has multiple dimensions like user role or language. You may want to pass some of these parameters in extra attributes or have the tag look them up in the FacesContext automatically. In this case an extension of the tag class would be sufficient. The extended tag class could overwrite the setProperties method and compile the automatically looked up dimensinos together with the ones provided in the additional attributes to form a more complicated EL expression.

### 6.1.3 Frames

When using frames with JSF there are two problems to solve:

- The request that reaches the HTML frameset is connected with a JSF state. If the request is redirected to 2 (or more) frames each of them will need its own JSF state. Concurrency problems when processing the 2 frames must be avoided.
- JSF requests need to have the request method `POST` but a HTML frameset redirects a request to its HTML frames using `HTTP GET`. This may include losing part of the state because the length of a HTTP `GET` is limited.

The solution includes implementation of new tags, UIComponents and renderers. The frameset will expose its JSF state and a serialized Map that contains all serializable request scope beans through JavaScript variables and redirect the frame request to a special servlet. Each frame is connected to a JSF action through an action attribute. The special servlet renders a HTML page that copies the JSF state from the framesets JavaScript variables to the frame and submits the frame with an HTTP POST. A filter will preprocess the requests of the special servlet, restore the JSF state and the request scope beans and trigger the frames action. The request is then forwarded to the action result according to the JSF naviation rules.

If you want to volunteer for adding fry:frameset and fry:frame tags to the J4Fry taglib, please contact me (Ganesh @J4Fry.org). I have already implemented these tags for a commercial client and thus solved the problems connected to the implementation.

### 6.1.4 Extend a beans scope

A bean with request scope may be needed in one more request, maybe to preserve input values from a former page that need to be redisplayed when a “back” button is pushed. You can achieve this by saving the bean properties in hidden input fields, but it is more convenient to have a tag that preserves a serializable bean in its component and restores it when decoding. This functionality requires implementation of tag, component and renderer (example: the Tomahawk t:saveState tag).

### 6.1.5 Trigger an action when loading a page

Before displaying a page an application may need to retrieve or prepare some data. You can implement the data retrieval in every action that forwards to the page, but it would be easier to have a tag that perform the retrieval action when loading the page. If the page is called initially you there is no action that forwards to the page and thus the action can only be called when the page is rendered.

To implement this the renderer would directly invoke the action in its encode method.

If you want to volunteer for adding a fry:onLoad tag to the J4Fry taglib, please contact me (Ganesh @J4Fry.org). I have already implemented this tag for a commercial client and thus solved the problems connected to the implementation.

### 6.1.6 Disable buttons after submit

A common problem in JSP programming is the double submit: The user clicks a submit button several times and thus sends several requests to the server. The server parallelly processes the requests thus reducing overall performance and maybe even running into concurrency problems. One clientside solution to this problem is to disable all submit buttons when one button is clicked.

An onClick event is supported by the standard JSF buttons and it could call a Javascript function that searches the page for submit buttons and disables them. This solution will perform poorly if the page contains many components.

An alternative are buttons that register themselves when rendered. As soon as one one them is clicked a JavaScript loop can explicitly disable all registered buttons. This solution requires implementation of a custom renderer that registers the buttons name.

### 6.1.7 Input fields, that don't submit the page when "enter" is pressed

HTML input fields submit the HTML form if the user presses the "enter" key while editing the form. You need to catch the fields keyPressed event and abort processing if the keyCode is 13 (=enter). You can do this with the standard JSF input field, but you have to repeat the code every time you need this functionality.

## 6.2 Taglibs (h, f, t, adf und rq)

Inside a JSP a taglib is referenced with an abbreviation the developer is free to define useing the @ taglib tag (see 2.7). Still there are conventions on the taglib abbreviations for the often used tag libraries.

These are the taglibs we are going to examine closer:

| Taglib | URI | Abbreviation |
|---|---|---|
| JSF HTML Tags | http://java.sun.com/jsf/html | H |
| JSF Core Tags | http://java.sun.com/jsf/core | F |
| MyFaces Extensions | http://myfaces.apache.org/tomahawk | T |
| Oracle ADF | http://xmlns.oracle.com/adf/faces | Af |

Examples of the JSF HTML and Core tags and the MyFaces Extensions (Tomahawk) can be viewed at:

http://www.irian.at/myfaces/home.jsf

## 6.3 JSF Core tags

The tags included in the JavaServer Faces core tag library are used to perform
core actions that are independent of a particular render kit. The tag descriptions are taken from the Java EE tutorial form SUN.

| **Tag** | **Functions** |
|---|---|
| actionListener | Registers an action listener on a parent component |
| phaseListener | Registers a PhaseListener instance on a UIViewRoot component |
| setPropertyActionListener | Registers a special action listener whose sole purpose is to push a value into a backing bean when a form is submitted |

| Tag | Functions |
|---|---|
| valueChangeListener | Registers a valuechange listener on a parent component |
| attribute | Adds configurable attributes to a parent component |
| converter | Registers an arbitrary converter on the parent component |
| convertDateTime | Registers a DateTime converter instance on the parent component |
| convertNumber | Registers a Number converter instance on the parent component |
| facet | Signifies a nested component that has a special relationship to its enclosing tag |
| loadBundle | Specifies a ResourceBundle that is exposed as a Map Parameter |
| param | Substitutes parameters into a MessageFormat instance and adds query string namevalue pairs to a URL |
| selectItem | Represents one item in a list of items in a UISelectOne or UISelectMany component |
| selectItems | Represents a set of items in a UISelectOne or UISelectMany component |
| subview | Contains all JavaServer Faces tags in a page that is included in another JSP page containing JavaServer Faces tags |
| validateDoubleRange | Registers a DoubleRangeValidator on a component |
| validateLength | Registers a LengthValidator on a component |
| validateLongRange | Registers a LongRangeValidator on a component |
| validator | Registers a custom validator on a component |
| verbatim | Generates a UIOutput component that gets its content from the body of this tag |
| View | Encloses all JavaServer Faces tags on the page |

## 6.4 JSF HTML tags

The tags included in the JavaServer Faces html tag library render HTML tags. The tag descriptions are taken from the Java EE tutorial from SUN.

| Tag | Functions | Rendered as | Appearance |
|---|---|---|---|
| Column | Represents a column of data in a UIData component. | A column of data in an HTML table | A column in a table commandButton |
| commandButton | Submits a form to the application. | An HTML <input type=*type*> element, where the *type* value can be submit, reset, or image | A button |
| commandLink | Links to another page or location on a page. | An HTML <a href> element | A hyperlink |
| dataTable | Represents a data wrapper. | An HTML <table> element | A table that can be updated dynamically |
| Form | Represents an input form. The inner tags of the form receive the data that will be submitted with the form. | An HTML <form> element | No appearance |
| graphicImage | Displays an image. | An HTML <img> element | An image |
| inputHidden | Allows a page author to include a hidden variable in a page. | An HTML <input type=hidden> element | No appearance |
| inputSecret | Allows a user to input a string without the actual string appearing in the field. | An HTML <inputtype=password> element | A text field, which displays a row of characters instead of the actual string entered |
| inputText | Allows a user to input a string. | An HTML <input type=text> element | A text field |

| Tag | Functions | Rendered as | Appearance |
|---|---|---|---|
| inputTextarea | Allows a user to enter a multiline string. | An HTML <textarea> element | A multirow text field |
| message | Displays a localized message. | An HTML  tag if styles are used | A text string |
| messages | Displays localized messages. | A set of HTML  tags if styles are used | A text string |
| outputFormat | Displays a localized message. | Plain text | Plain text |
| outputLabel | Displays a nested component as a label for a specified input field. | An HTML <label> element | Plain text |
| outputLink | Links to another page or location on a page without generating an action event. | An HTML <a> element | A hyperlink |
| outputText | Displays a line of text. | Plain text | Plain text |
| panelGrid | Displays a table. | An HTML <table> element with <tr> and <td> elements | A table |
| panelGroup | Groups a set of components under one parent. | | A row in a table |
| selectBooleanCheckbox | Allows a user to change the value of a Boolean choice. | An HTML <input type=checkbox> element. | A checkbox |
| selectItem | Represents one item in a list of items in a UISelectOne component. | An HTML <option> element | No appearance |
| selectItems | Represents a list of items in a UISelectOne component. | A list of HTML <option> elements | No appearance |
| selectManyCheckbox | Displays a set of checkboxes from which the user can select multiple values. | A set of HTML <input> elements of type checkbox | A set of checkboxes |
| selectMany | | | |
| listbox | Allows a user to select multiple items from a set of items, all displayed at once. | An HTML <select> element | A list box |
| selectManyMenu | Allows a user to select multiple items from a set of items. | An HTML <select> element | A scrollable combobox |
| selectOneListbox | Allows a user to select one item from a set of items, all displayed at once. | An HTML <select> element | A list box |
| selectOneMenu | Allows a user to select one item from a set of items. | An HTML <select> element | A scrollable combobox |
| selectOneRadio | Allows a user to select one item from a set of items. | An HTML <input type=radio> element | A set of radio buttons |

## 6.5 Tomahawk tags

The MyFaces distribution comes with the Tomahawk component that enhance JSF functionality.

Most Tomahawk tags require the MyFaces Extensions filter to be installed. Install the filter by adding it to your web.xml. You need 2 mappings, because some tags address resources through the second mapping:

```
<filter>
      <filter-name>MyFacesExtensionsFilter</filter-name>
      <filter-class>
```

```
                org.apache.myfaces.webapp.filter.ExtensionsFilter
        </filter-class>
        <init-param>
                <param-name>uploadMaxFileSize</param-name>
                <param-value>20m</param-value>
        </init-param>
        <init-param>
                <param-name>uploadThresholdSize</param-name>
                <param-value>100k</param-value>
        </init-param>
        <init-param>
                <param-name>uploadRepositoryPath</param-name>
                <param-value>c:\projects\system</param-value>
        </init-param>
</filter>

<filter-mapping>
        <filter-name>MyFacesExtensionsFilter</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>

<filter-mapping>
        <filter-name>MyFacesExtensionsFilter</filter-name>
        <url-pattern>/faces/myFacesExtensionResource/*</url-pattern>
</filter-mapping>
```

You can find the explanations of this table and the detailed explanation how each tags attributes are used in the `myfaces-1.1.1/tlddoc/tomahawk` folder.

| Tag | Functions |
|---|---|
| aliasBean | A tag that defines a new bean (alias) with a given value. This allows you to design a subform with a generic (fictive) beans and to include it in all the pages where you use it. You just need to make an alias to the real bean named after the generic bean before including the subform. When used within an aliasBeansScope tag, this tag adds the alias to the aliasBeansScope. This makes configuration with multiple aliasBeans easier to write. |
| aliasBeansScope | This is like an aliasBean tag, but instead of the alias/value attributes, you configure the aliases by adding aliasBean tags in the body. The aliasBeans should be declared right after this tag. |
| Buffer | Renders a HTML input of type "checkbox". The associated SelectItem comes from an extended selectManyCheckbox component with layout "spread". The selectManyCheckbox is referenced by the "for" attribute. All HTML pass-through attributes for this input are taken from the associated selectManyCheckbox |
| Checkbox | Renders a HTML input of type "checkbox". The associated SelectItem comes from an extended selectManyCheckbox component with layout "spread". The selectManyCheckbox is referenced by the "for" attribute. All HTML pass-through attributes for this input are taken from the associated selectManyCheckbox. |
| collapsiblePanel | |
| Column | |
| Columns | |
| commandButton | Extends standard commandButton by user role support. |
| commandLink | Extends standard commandLink by user role support and the HTML target attribute. |
| commandNavigation | Must be nested inside a panel_navigation action and renders a clickable navigation item. This action is derived from the standard command_link action and has equal attributes. (Replaces former "navigation_item" tag.) |

| Tag | Functions |
|---|---|
| commandNavigation2 | Must be nested inside a panel_navigation action and renders a clickable navigation item. This action is derived from the standard command_link action and has equal attributes. (Replaces former "navigation_item" tag.) |
| commandSortHeader | Clickable sort column header. Must be nested inside an extended data_table tag. This tag is derived from the standard command_link tag and has the additional attributes columnName and arrow. Note: In contrary to normal command links the default for the "immediate" attribute is "true". |
| dataList | Similar to dataTable, but does not render a table. Instead the layout attribute controls how each dataRow is rendered. |
| dataScroller | Scroller for UIData components eg. dataTable Must be nested inside footer facet of dataTable OR for attribute must be given so that corresponding uiData can be found |
| dataTable | Extended data_table that adds some additional features to the standard data_table action: see attribute descriptions for preserveDataModel, sortColumn, sortAscending and preserveSort. |
| Div | Places a div around its children |
| graphicImage | Extends standard graphicImage. |
| htmlTag | Places the given HTML tag div around its children |
| iconProvider | |
| inputCalendar | Provides a calendar. |
| inputDate | |
| inputFileUpload | You must enable the MultiPart Filter to make this component work (see web.xml). Also, don't forget to set the form's attribute "enctype" to "multipart/form-data". See "examples/web/fileupload.jsp" for an example! |
| inputHidden | Extends standard inputHidden by providing additional MyFaces goodies. |
| inputHtml | An inline HTML based word processor based on the Kupu library. See http://kupu.oscom.org Right now, the support is limited to one editor per page (but you can use tabs to have multiple editors, but only one rendered at a time). |
| inputSecret | Extends standard inputSecret. |
| inputText | Extends standard inputText by user role support. |
| inputTextarea | Extends standard inputTextarea by user role support. |
| inputTextHelp | Extends standard inputText by helptext support. |
| jscookMenu | |
| jsValueChangeListener | Value change listener on client side. |
| jsValueSet | Setting a value from the model in java-script so that it can be used (e.g. by the value change listener) afterwards. |
| Message | MyFaces extension to the standard messages tag: see summaryDetailSeparator attribute |
| Messages | MyFaces extension to the standard messages tag: see showInputLabel attribute |
| navigationMenuItem | |
| navigationMenuItems | |
| newspaperTable | A data table for rendering long skinny tables as short wide table by wrapping the table over a specified number of columns. |
| outputLabel | Extends standard outputLabel by user role support. |
| outputText | Extends standard outputText by user role support. |
| panelGrid | Extends standard panelGrid by user role support. |
| panelGroup | Extends standard panelGroup by user role support. |
| panelLayout | |
| panelNavigation | |
| panelNavigation2 | |
| panelStack | |
| panelTab | |
| panelTabbedPane | |

| Tag | Functions |
|---|---|
| Popup | Renders a popup which displays on a mouse event. |
| Radio | Renders a HTML input of type "radio". The associated SelectItem comes from an extended selectOneRadio component with layout "spread". The selectOneRadio is referenced by the "for" attribute. All HTML pass-through attributes for this input are taken from the associated selectOneRadio. |
| saveState | saveState enables you to persist beans and values longer than request scope, but shorter than session scope. It is ideal for persisting backing beans and values with the same scope as your view components. It does this by saving the target state with the component tree. |
| selectBooleanCheckbox | Extends standard selectBooleanCheckbox by user role support. |
| selectManyCheckbox | Extends standard selectManyCheckbox by user role support. Additionally this extended selectManyCheckbox accepts a layout attribute of value "spread" (see custom checkbox tag). |
| selectManyListbox | Extends standard selectManyListbox by user role support. |
| selectManyMenu | Extends standard selectManyMenu by user role support. |
| selectOneCountry | A localized list of countries choose box. The value binds to the country ISO 3166 code. This is the same code as for java.util.Locale.getCountry(). The official codes list is available here : http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html |
| selectOneLanguage | A localized list of languages choose box. The value binds to the language ISO 639 code (lowercase). This is the same code as for java.util.Locale.getLanguage(). The official codes list is available here : http://www.loc.gov/standards/iso639-2/englangn.html |
| selectOneListbox | Extends standard selectOneListbox by user role support. |
| selectOneMenu | Extends standard selectOneMenu by user role support. |
| selectOneRadio | Extends standard selectOneRadio by user role support. Additionally this extended selectOneRadio accepts a layout attribute of value "spread" (see custom radio tag). |
| Stylesheet | a component, which renders the path to a common CSS-file |
| swapImage | |
| tabChangeListener | |
| Tree | |
| tree2 | |
| treeCheckbox | Renders a HTML input of type "treeCheckbox". The associated comes from the treeCheckbox itemLabel and itemValue. The selected items come from an extended selectManyCheckbox component with layout "spread". The selectManyCheckbox is referenced by the "for" attribute. All HTML pass-through attributes for this input are taken from the associated selectManyCheckbox. |
| treeColumn | Renders a HTML input of type "treeColumn". This tag outlines the column where the tree structure will be render as part of the tree table. |
| treeSelectionListener | |
| updateActionListener | Registers a org.apache.myfaces.custom.updateactionlistener.UpdateActionListener at the parent component. |
| validateCreditCard | A custom validator for creditCards, based upons Jakarta Commons. |
| ValidateEmail | A custom validator for email address format, based upons Jakarta Commons. |
| ValidateEqual | A custom validator for validations against foreign component values. |
| validateRegExpr | A custom validator for reg. expr., based upons Jakarta Commons. |

## 6.6 ADF tags

The ADF Tag library was originally developed by Oracle. Oracle had an older library named ADF UIX that was migrated to Oracle ADF based on JSF. It was donated to the Apache Software Foundation and contains nearly 100 components. The documentation of the tags with graphical examples is available at:

http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/doc/index.html#Documentation