

# Easy Java Websites (EJW)

## Configuration

An Extremely Easy To Use  
Web Application Development Framework  
for Java

Copyright © 2005 - Present, EasierJava All Rights Reserved.

This Easier Java Persistence manual is a copyrighted work and is not transferable. If you received this manual somewhere other than EasierJava.com, please report it.

EasierJava and EJW are trademarks or registered trademarks of EasierJava, Inc. in the U.S. and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All other marks are trademarks or registered trademarks of their respective holders in the U.S. and other countries.

<b>EASY JAVA WEBSITES CONFIGURATION.....</b>	<b>4</b>
REQUESTS.....	4
<i>Parameters.....</i>	6
<i>Security Roles.....</i>	6
FORWARDING.....	6
<i>Forward.....</i>	7
<i>Forward References.....</i>	7
REDIRECTS.....	8
<i>Redirect.....</i>	8
<i>Redirect References.....</i>	9
URL.....	9
<i>Special Forward/Redirect Conditions.....</i>	10
<i>URL Referencing.....</i>	10
DATA VALIDATION.....	11
<i>Validation.....</i>	11
<i>Validation References.....</i>	12
MESSAGES.....	13
<b>CONFIGURING THE EJW SERVLET .....</b>	<b>13</b>
SERVLET 3.X ANNOTATIONS.....	13
SERVLET DEPLOYMENT DESCRIPTOR (WEB.XML) .....	14
<i>Servlet (required).....</i>	15
<i>configFile (optional).....</i>	15
<i>useDomainName (optional).....</i>	15
<i>validateXML (optional).....</i>	16
<i>initDestroyHandler (optional).....</i>	16
<i>classPrefix (optional).....</i>	17
<i>urlRewriting (optional).....</i>	17
<i>Servlet Mapping (required).....</i>	17
<b>SPECIAL TOPICS .....</b>	<b>18</b>
LOGGING.....	18
REST (RESTFUL WEB SERVICE) REQUESTS.....	18
AJAX REQUESTS.....	20
DEFAULT REQUEST.....	20
ERROR HANDLING.....	21
SECURITY/USER AUTHENTICATION.....	21
SECURE CONNECTIONS.....	22
URL REWRITING.....	23
<b>APPENDIX A – VALIDATION.....</b>	<b>23</b>
ENDSWITH.....	24
ENDSWITHIGNORECASE.....	24
ISBYTE.....	24
ISCREDITCARD.....	24
ISDATE.....	24
ISDOUBLE.....	25
ISEMAIL.....	25
ISEQUALTO.....	25
ISEQUALTOIGNORECASE.....	25
ISFLOAT.....	25
ISGREATEROREQUALTOMIN.....	25
ISGREATEROREQUALTOMINLENGTH.....	26
ISINRANGE.....	26
ISINT.....	26

<u>ISISBN</u> .....	<u>26</u>
<u>ISLESSOREQUALTOMAX</u> .....	<u>26</u>
<u>ISLESSOREQUALTOMAXLENGTH</u> .....	<u>26</u>
<u>ISLONG</u> .....	<u>27</u>
<u>ISNOTEMPTYORNULL</u> .....	<u>27</u>
<u>ISEMPTYORNULL</u> .....	<u>27</u>
<u>ISHORT</u> .....	<u>27</u>
<u>ISSTRING</u> .....	<u>27</u>
<u>ISSTRINGIGNORECASE</u> .....	<u>27</u>
<u>ISTRUEFALSE</u> .....	<u>27</u>
<u>ISURL</u> .....	<u>28</u>
<u>ISYESNO</u> .....	<u>28</u>
<u>MATCHESREGEXP</u> .....	<u>28</u>
<u>STARTSWITH</u> .....	<u>28</u>
<u>STARTSWITHIGNORECASE</u> .....	<u>28</u>
<b><u>APPENDIX B – EJW DTD</u></b> .....	<b><u>28</u></b>

# Easy Java Websites Configuration

NOTE: With configuration based requests everything else is the same as configuration free requests (see configuration free documentation).

EJW provides a completely optional, but simple, XML configuration mode which consists of:

```
<EJW>
  <request id="" match="" matchRegExp="" secureConnection=""
    handlerClass="" handlerMethod="">
    <security role="" />
    <parameter name="" value="" />
    <forward id="" key="" protocol="" url="" rewrite="" />
    <redirect id="" key="" protocol="" url="" rewrite="" />
    <validation id="" parameterName="" type="" arguments="" errorMessage="" />
  </request>

  <url id="" protocol="" url="" rewrite="" />
  <message id="" value="" />
</EJW>
```

This mode provides the ability for class and method reuse, passing in static parameters, regular expression matching of URIs, soft coded URIs, and soft coded messages. EJW configuration even allows overriding configuration free and other configurations. The configuration mode also provides for XML defined security, forwarding, redirects and validation.

## Requests

Requests allow you to associate a piece of Java executable code to a web request. All parameters, security roles, forwards, redirects, and validations are used in the context of a request, and a request can have any number of each.

### Syntax:

```
<request id="" match="" matchRegExp="" secureConnection=""
  handlerClass="" handlerMethod="" />
```

Id	The request Id is equal to the value returned by request.getRequestURI(), with all leading path information removed, leaving only the filename portion of the URI. The request Id must also be
----	--

	<p>unique within the document. For example:</p> <pre> /webapp/p1/customer /webapp/p1/customer/customer.ejw </pre> <p>would produce the following request Ids:</p> <pre> customer customer.ejw </pre> <p>This value is your request Id.</p>
match	<p>The match attribute overrides matching the request Id to the URI and will instead match the request URI to the match string. Can use characters that can't be used in an XML Id. This attribute is optional and overrides matching the request Id.</p>
matchRegExp	<p>The matchRegExp attribute overrides both matching the request Id and the match string and will instead match the request URI to the regular expression string. Great flexibility in matching URIs and you can use wildcards the regex way(*). This attribute is optional and overrides matching both the request Id and match.</p>
secureConnection	<p>If true, the request will only be processed over a secure (https) connection. This attribute is optional and the default is false.</p>
handlerClass	<p>The Java class instance to associate with the request. This class must extend RequestHandler. If this attribute is not defined, then there must be a forward with 'key="success"' defined or a "home" url must be defined.</p>
handlerMethod	<p>The method name in the class that is to be called. The method defined in the class should have "validate" prepended for a validation method. But, "validate" shouldn't be included in the "handlerMethod" definition. During a web request, EJW will check the class for a method matching the name "validate[Method]()", if found it will be called.</p> <p>If handlerMethod is not defined, the default name "request" will be used and the same process above will be used with validateRequest() being called first (if it exists), followed by request().</p>

## Parameters

```
<parameter name="" value="" />
```

Parameters define information that can be passed to and accessed by your RequestHandler classes.

name	a key value associated with value
value	The data value of interest.

Accessing parameter values within code is done with:

```
ServerInterface.getRequestData().getParameter("parameterName");
```

## Security Roles

This element is optional, but when defined, the request will be limited to authenticated users who have the defined role associated with them. There can be any number of security roles defined for a given request. Requests that don't have any roles defined are not limited, and can be accessed by any authenticated or non-authenticated user. A user only has to be in one role to access the request.

If a non-authenticated user or a user that does not possess the role tries to access a request limited by roles, EJW will throw an error and return the forwarding indicator "authenticationError". If no forwarding is defined in the request for "authenticationError", EJW will fall back to global "error" forwarding.

### Syntax:

```
<security role="" />
```

role	A valid security role associated with authenticated users.
------	--

## Forwarding

Request forwarding is central to web application flow control and goes hand-in-hand with your RequestHandler's methods. The string values your RequestHandler methods return have to exist in your request forwarding as a forward or a forwardReference definition that points to a defined forward. A request can have any number of forwards and forward references defined.

Note: urls, Forwards, and redirects contain the same information and can be used interchangeably when being referenced.

## Forward

### Syntax:

```
<forward id="" key="" protocol="" url="" rewriteUrl="" />
```

Id	Id is a valid XML Id string that is unique within the document.
Key	<p>Key is usually a value returned by your RequestHandler methods (see Special Forwarding Conditions) and is defined by you for you. Your RequestHandler validation method should return “success”, “validationError”, or another value that is defined in your request forwarding.</p> <p>Your RequestHandler processing method can return anything that is defined in your request forwarding or “noForwarding” if the request has been satisfied, and no return page is required.</p>
Protocol	<p>This attribute is optional and useful for forcing a switch between protocols (i.e., http to https, https to http) in relative URLs.</p> <p>Protocol is only useful when referencing forwarding URLs in web authoring via “<code>{ejwUrl.id}</code>” for menus, form actions, and other URL-linking purposes.</p>
url	Any valid complete or context relative URL.
rewriteUrl	True or False. Use URL rewriting for cookie limited browsers. Defaults to true.

## Forward References

“forwardReference” references a defined forward. You can use “forwardReference” inside any number of requests to use a forward over and over. The key can be different with any of the forward references.

### Syntax:

```
<forwardReference idRef="" key="" />
```

idRef	idRef is a valid XML Id string that references a forward definition that is defined within the document.
Key	Key is usually a value returned by your RequestHandler methods (see Special Forwarding Conditions) and is defined by you for you. Your RequestHandler validation method should return “success”, “validationError”, or another value

	<p>that is defined in your request forwarding.</p> <p>Your RequestHandler processing method can return anything that is defined in your request forwarding or “noForwarding” if the request has been satisfied, and no return page is required.</p>
--	---

## Redirects

Redirects are useful for redirecting a browser to a different URL. You can use this feature to send a user to a secure connection when coming in over an insecure connection. Or use it simply to redirect to a different page.

Note: urls, Forwards, and redirects contain the same information and can be used interchangeably when being referenced.

## Redirect

### Syntax:

```
<redirect id="" key="" protocol="" url="" rewriteUrl="" />
```

id	Id is a valid XML Id string that is unique within the document.
Key	<p>Key is usually a value returned by your RequestHandler methods (see Special Forwarding Conditions) and is defined by you for you. Your RequestHandler validation method should return “success”, “validationError”, or another value that is defined in your request forwarding.</p> <p>Your RequestHandler processing method can return anything that is defined in your request forwarding or “noForwarding” if the request has been satisfied, and no return page is required.</p> <p>This attribute is not needed with globally scoped forwards.</p>
Protocol	This attribute is optional and useful for forcing a switch between protocols (i.e. http to https, https to http) in relative URLs.
url	Any valid complete or context relative URL.
rewriteUrl	True or False. Uses URL rewriting for cookie limited browsers. Defaults to true.

## Redirect References

“redirectReference” references a defined redirect. You can use “redirectReference” inside any number of requests to use a redirect over and over. The key can be different with any of the redirect references.

### Syntax:

```
<redirectReference idRef="" key="" />
```

idRef	idRef is a valid XML Id string that references a redirect definition that is defined within the document.
Key	Key is usually a value returned by your RequestHandler methods (see Special Forwarding Conditions) and is defined by you for you. Your RequestHandler validation method should return “success”, “validationError”, or another value that is defined in your request forwarding.  Your RequestHandler processing method can return anything that is defined in your request forwarding or “noForwarding” if the request has been satisfied, and no return page is required.

## *url*

url is the global version of forward and redirect and is the only URL information that can be defined outside a request. Like forwards and redirects it can be referenced by either forwardReference or redirectReference from within a request. url is provided as a more intuitive version of the forward statement to coincide with the `{ejwUrl}` for page authoring.

### Syntax:

```
<url id="" protocol="" url="" rewriteUrl="" />
```

Id	Id is a valid XML Id string that is unique within the document.
Protocol	This attribute is optional and useful for forcing a switch between protocols (i.e. http to https, https to http) in relative URLs.  Protocol is only useful when referencing forwarding URLs in web authoring via “ <code>{ejwUrl.id}</code> ” for menus, form actions, and other URL-linking purposes.  Protocol is the value inserted before the beginning of the

	URL and can be any valid URL type (i.e. http, https, etc.).
url	Any valid complete or context relative URL.
rewriteUrl	True or False. Use URL rewriting for cookie limited browsers. Defaults to true.

## Special Forward/Redirect Conditions

Certain situations call for special forward/redirect handling. The following lists the special cases defined for EJW that your RequestHandler methods can return, and with the exception of “noForwarding”, your forward/redirect definitions can catch these special conditions by defining the following as forward keys for a request.

<b>noForwarding</b>	Request processing stops. Useful for handling web events that don't need a page returned.
---------------------	---

<b>authenticationError</b>	This forward key is returned when there is no user logged in or the user does not have permission for the request.
<b>notSecureError</b>	This forward key is returned when a request requires a secure connection but the connection is not secure.
<b>validationError</b>	A validation error was encountered.
<b>login</b>	In the case of an authentication error, if the authenticationError key is not defined, EJW will automatically forward to a login page if one is defined. Otherwise, EJW will forward to the error page.
<b>error</b>	Any errors that may occur will be forwarded to the defined error page.
<b>home</b>	The web applications home page.

## URL Referencing

URLs can have references to other urls, forwards, and redirects by using:

“`${ejwUrl/id}`”

which will include the URL, as is, into the URL doing the referencing. For example:

```
<forward id="main" url="/mainView.jsp" />
```

with:

```
<forward id="home" protocol="http" url="${main}?c=intro" />
```

will result in a URL that equals “/mainView.jsp?c=intro”.

There can be any number of references in a URL, and URLs can reference a URL which also has a URL reference. However, a URL reference must have ‘\${’, followed by a ‘forward/redirect id’, followed by ‘}’.

## Data Validation

Data validation can be handled from two places. One is from your Java code. The other is automatic validation from validation definitions in your XML configuration file.

You can access the validation handler from your Java code via `ServerInterface.getValidationHandler()`, which returns a reference to a `ValidationHandler` that contains methods corresponding to all the methods available with XML-based validation.

Using “validation” and “validationReference”, you can define any number of validations for your incoming data.

A data element can have any number of validations, as well. You can first check for “isNotEmptyOrNull” and then check if it is a properly formatted email with “isEmail”.

**NOTE: Empty data is not considered an error by default, so you must check specifically for an empty or null value using “isNotEmptyOrNull”, as validations do not consider it an error if the value is empty, null, or not found.**

## Validation

Validations are defined globally and then referenced in requests via “validationReference”. A request can have any number of validations defined for the incoming data parameters.

### Syntax:

```
<validation id="" parameterName="" type=""  
            arguments="" errorMessage=""/>
```

id	Id is a valid XML Id string that is unique within the document.
parameterName	The form/query string parameter name, etc., as returned by <code>ServletRequest.getParameter(parameterName)</code> .
type	Defines the type of data being validated, and is one of:  endsWith endsWithIgnoreCase isByte isCreditCard isDate

	isDouble isEmail isEqualTo isEqualToIgnoreCase isFloat isGreaterThanOrEqualToMin isGreaterThanOrEqualToMinLength isInRange isInt isIsbn isLessThanOrEqualToMax isLessThanOrEqualToMaxLength isLong isNotEmptyOrNull isShort isSubstring isSubstringIgnoreCase isTrueFalse isUrl isYesNo matchesRegExp startsWith startsWithIgnoreCase
	See Appendix A for details on validation types.
arguments	Some validations, such as comparison validations, require an argument, such as the string to compare with. This attribute is usually not needed, but depends on the needs of the validation.
errorMessage	The message you want to display to the user.

## Validation References

“validationReference” references a defined validation. You can set up a global validation, and then use “validationReference” inside any number of requests to use the validation over and over. The parameter name can be different with any of the validation references.

### Syntax:

```
<validationReference idRef="" parameterName="" />
```

idRef	idRef is a valid XML Id string that references a validation definition that is defined elsewhere within the document.
parameterName	The form/query string parameter name, etc., as returned

	by <code>ServletRequest.getParameter(parameterName)</code> .
--	--

## Messages

Messaging is useful for allowing “soft” coded messages that can be changed anytime without going into the web page source. You can set up any number of messages and access them via:

```
{ejwMessage.id}
```

in your web page. Messaging does not currently support “`java.util.Locale`” directly. However, JSTL/JSP does, and it’s not hard to set up in Velocity.

### Syntax:

```
<message id="" value="" />
```

id	Id is a valid XML Id string that is unique within the document.
value	The message to display to the user via “ <code>{ejwMessage.id}</code> ” in the view.

## Configuring the EJW Servlet

EJW supports both servlet 3.x annotations, as well as, typical deployment discriptor (`web.xml`) definitions.

### Servlet 3.x Annotations

NOTE: As you well know, annotations are equal to hard coding, so proceed with that knowledge.

By implementing the following:

```
@WebServlet(urlPatterns={"/"},
    initParams={
        @WebInitParam(name="param-name", value="param-value")
    })
```

```
public class SimpleController extends RequestServlet { }
```

you can create your own custom controller (EJW handles controller functionality automatically) to take advantage of Servlet 3.x annotations. This allows you to do away with the web.xml deployment descriptor. You can use any of the Servlet 3.x annotations.

You can also use the `@MultipartConfig` annotation for file uploads. However, EJW also supports file uploads and both the Servlet 3.x implementation and EJW use Apache's commons file upload package. The difference is EJW handles multipart form data requests dynamically, where as, the servlet spec 3.x requires the request to be multi-part form data.

## ***Servlet Deployment Descriptor (web.xml)***

The following Tomcat example is based on the standard Servlet specification 2.4, which you can download from Sun Microsystems' Java website at <http://java.sun.com>.

You can define the servlet with "RequestServlet" or any other name you like, which you will map requests to a little later. The servlet supports several initialization parameters, which are defined below.

```
<servlet>
  <servlet-name>RequestServlet</servlet-name>
  <servlet-class>ejw.RequestServlet</servlet-class>
  <init-param>
    <param-name>configFile</param-name>
    <param-value>/WEB-INF/webapp.xml</param-value>
  </init-param>
  <init-param>
    <param-name>useDomainName</param-name>
    <param-value>your_host_name</param-value>
  </init-param>
  <init-param>
    <param-name>validateXml</param-name>
    <param-value>>true</param-value>
  </init-param>
  <init-param>
    <param-name>initDestroyHandler</param-name>
    <param-value>business.AppUtils</param-value>
  </init-param>
  <init-param>
    <param-name>classPrefix</param-name>
    <param-value>webapp</param-value>
  </init-param>
  <init-param>
    <param-name>urlRewriting</param-name>
    <param-value>>true</param-value>
  </init-param>
```

```

    <init-param>
      <param-name>handleMimeTypes</param-name>
      <param-value>jpg, swf</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>RequestServlet</servlet-name>
    <url-pattern>*.ejw</url-pattern>
  </servlet-mapping>

```

## Servlet (required)

The servlet class is always “ejw.RequestServlet” and is defined with:

```

<servlet>
  <servlet-name>RequestServlet</servlet-name>
  <servlet-class>ejw.RequestServlet</servlet-class>
</servlet>

```

where “servlet-name” is anything you like, and will be referenced later in the servlet-mapping definition.

## configFile (optional)

The EJW configuration file is an XML file (defined later) used to define requests, forwarding, validation, etc. EJW uses `ServletContext.getResource()` to locate the file, so the file can be located anywhere that `ServletContext.getResource()` can find a resource. The best place for it is probably in “/WEB-INF”.

Example:

```

<init-param>
  <param-name>configFile</param-name>
  <param-value>/WEB-INF/webapp.xml</param-value>
</init-param>

```

## useDomainName (optional)

Used with page authoring and URL building via `forwardingUrlBuilder.forwardId`, which allows you to force multiple domains into one default domain. This could be useful in situations like security certificates, where you can only have a single certificate defined, but multiple domains are mapped to a single IP. The default is for `forwardingUrlBuilder.forwardId` to use

the domain returned by `ServletRequest.getServerName()`, which is the domain the user typed in the browser's address bar.

Example:

```
<init-param>
  <param-name>useDomainName</param-name>
  <param-value>www.softwaresensation.com</param-value>
</init-param>
```

### **validateXML (optional)**

By default, EJW does not use a validating parser while parsing the XML configuration. It is best to use this entry with “true” defined to make sure the configuration is conforming to the XML standard and to pick up any DTD-related errors that may be present.

Example:

```
<init-param>
  <param-name>validateXml</param-name>
  <param-value>true</param-value>
</init-param>
```

### **initDestroyHandler (optional)**

In the case where you have startup and/or cleanup needs with your web application, EJW allows you to define a class to handle servlet startup and shutdown events. This class is defined to EJW with:

```
<init-param>
  <param-name>initDestroyHandler</param-name>
  <param-value>java_class_without_extension</param-value>
</init-param>
```

The class must extend the class `InitDestroyHandler`:

```
public class InitDestroyHandler
{
  public void init(ServletContext context) throws Exception {}
  public void destroy(ServletContext context) throws Exception {}
}
```

The “init” method will be called with the servlets first use and the “destroy” method will be called when the servlet is shut down.

Example:

```
<init-param>
  <param-name>initDestroyHandler</param-name>
  <param-value>business.AppUtils</param-value>
</init-param>
```

### **classPrefix (optional)**

The class prefix is used for configuration free requests and is simply prepended to the URI. This provides a secure method for locating classes dynamically.

Example:

```
<init-param>
  <param-name>classPrefix</param-name>
  <param-value>webapp</param-value>
</init-param>
```

### **urlRewriting (optional)**

By default, EJW performs URL rewriting for  $\${ejwUrl.key}$ . If you do not want URL rewriting, you can turn it off with:

```
<init-param>
  <param-name>urlRewriting</param-name>
  <param-value>>false</param-value>
</init-param>
```

### **handleMimeTypes (optional)**

By default EJW ignores mime typed requests (i.e. gif, jpg, swf, etc.) and simply passes them on to be handled as normal. If you want to handle these requests, you'll need to define the mime types with the following:

```
<init-param>
  <param-name>handleMimeTypes</param-name>
  <param-value>gif, jpg</param-value>
</init-param>
```

For example, you might want to do this when implementing an image server.

## Servlet Mapping (required)

Servlet mapping is part of the servlet specification 2.4, which you can download from Sun Microsystems' Java website at <http://java.sun.com>. Servlet mapping goes hand-in-hand with the servlet definition above. You can define any number of mappings, and you can map any request pattern to the servlet.

Example:

```
<servlet-mapping>
  <servlet-name>RequestServlet</servlet-name>
  <url-pattern>/webapp/*</url-pattern>
</servlet-mapping>
```

Usual mappings are path-oriented mappings:

```
/path/*
/path/exact
```

or extension-oriented mappings:

```
*.ext
```

## Special Topics

### *Logging*

NOTE: With logging set to DEBUG, you can see diagnostic information including details of the web requests.

EJW uses slf4j ([www.slf4j.org](http://www.slf4j.org)). Slf4j is a simple logger facade 4 Java. It supports the major logging frameworks such as log4j, Java logging, logback, commons logging, etc.

slf4j (required):

Download from: [www.slf4j.org](http://www.slf4j.org)  
Must have slf4j-api.jar in your classpath.

slf4j simple (defaults to info):

Download from: [www.slf4j.org](http://www.slf4j.org)  
Copy slf4j-api.jar and slf4j-simple.jar to your classpath.

logback (defaults to debug):

Download from: [logback.qos.ch](http://logback.qos.ch)

Copy slf4j-api.jar, logback-classic.jar and logback-core.jar to your classpath.

log4j (requires configuration):

Download from: <http://logging.apache.org/log4j>

Copy slf4j-api.jar, slf4j-log4j.jar and log4j.jar to your classpath.

## ***REST (RESTful Web Service) Requests***

NOTE: Regular EJW request handling can also be RESTful, as the server interface will provide any information required to figure out what type of request is made, as well as, the path arguments sent.

RESTful requests are handled in a configuration free manner. The request is matched to a class:

```
public class MyRestfulHandler extends RestfulHandler
{
    public String get(ServerInterface serverInterface, String[] args) { ... }
    public String post(ServerInterface serverInterface, String[] args) { ... }
    ...
}
```

with an exact match of what is defined in the servlet mapping (the servlet path):

```
<servlet-mapping>
  <servlet-name>ejwRequestServlet</servlet-name>
  <url-pattern>/myRestfulHandler</url-pattern>
  <url-pattern>/myRestfulHandler/*</url-pattern>
</servlet-mapping>
```

Any additional path information is parsed and passed in to the request handler as a String array (String[] args).

With this definition, EJW can find any request handler for any supported URI and any supported HTTP method.

Within your request handler methods you can handle input with request.getInputStream() or request.getReader() and output with response.getOutputStream() or response.getWriter(). In this case you will want to return NO\_FORWARDING or super.method(...).

However, it would be a huge waste to limit your self to this mode of input and output. You can still handle URL encoded data sent to the server and access it with getParameter().

More importantly you can return a forwarding URI to a template rendering engine such as JSP/JSTL or Velocity.

For example, if your RESTful GET request returns plain text with variable data, you can use:

```
<%@ page contentType="text/plain" %>
```

```
FirstName = ${contact.firstName}
```

```
LastName = ${contact.lastName}
```

```
Address = ${ contact.address}
```

```
...
```

or return a JSON formatted document for your JavaScript:

```
<%@ page contentType="text/plain" %>
```

```
// JSON formatted date
```

```
{"dayOfWeek" : "${date.dayOfWeek}",
```

```
"dayOfMonth" : "${date.dayOfMonth}",
```

```
"month" : "${date.month}",
```

```
"year" : "${date.year}",
```

```
"hour" : "${date.hour}",
```

```
"minute" : "${date.minute}",
```

```
"second" : "${date.second}",
```

```
"message" : "(Click for update)" }
```

and, you can just as easily handle XML, HTML, etc. based responses.

## ***AJAX Requests***

Handling AJAX requests is completely trivial. Your client can access any HTTP oriented request/response (with any HTTP method) that you set up with EJW in any mode (configuration free, RESTful, and/or configuration based).

And as described above, you can return a forwarding URI to a template rendering engine such as JSP/JSTL or Velocity:

```
<%@ page contentType="text/plain" %>
```

```
// JSON formatted date
```

```
{"dayOfWeek" : "${date.dayOfWeek}",
```

```
"dayOfMonth" : "${date.dayOfMonth}",
```

```
"month" : "${date.month}",  
"year" : "${date.year}",  
"hour" : "${date.hour}",  
"minute" : "${date.minute}",  
"second" : "${date.second}",  
"message" : "(Click for update)" }
```

and you can just as easily handle XML, HTML, etc. based responses.

## ***Default Request***

In both configuration free and configuration based requests, there can be defined a default request that performs certain actions when no request is found to handle a requestURI. The default request is simply named default and can be defined in the same way:

```
webapp.Default.class
```

for configuration free. And:

```
<request id="Default.ejw" handlerClass="webapp.Default" />
```

for configuration based.

Usually the default request handler will simply forward all requests to the website's home page.

## ***Error Handling***

Error handling with configuration is handled with the following forwarding keys:

```
authenticationError  
notSecureError  
validationError  
error
```

And you simply handle them with forwarding/redirects. See Special Forward/Redirect Conditions above.

For configuration free, any exceptions thrown back to the EJW servlet will be sent to /WEB-INF/error.jsp.

An all situations, an exception that is thrown back to the EJW servlet will be available to page rendering in `{ejwException}`, and a normalized error message will be available in `{errorMessage}`.

## ***Security/User Authentication***

EJW security is all about restricting requests based on security roles. EJW has built-in support for both container-based and application-based security.

If using container-based security, EJW automatically handles it and nothing further is required. EJW will use the `getRemoteUser()` and `isUserInRole()` methods defined in the `ServletRequest` class.

However, application-based security can flow more smoothly when the user is logging in and out. It is best done over a secure connection, but this is not required. If using application-based security, you will need to handle the login and call the `ServerInterface` method:

```
setRemoteUserInformation()
```

with the following arguments:

```
String user_id,  
String salutation,  
String first_name,  
String last_name,  
HashSet roles
```

“salutation”, “first\_name” and “last\_name” can be null and “roles” is just a `HashSet` of strings. From that point, EJW handles security the same way it would for container-based security.

Logging the user out can be done with:

```
ServerInterface.invalidateRemoteUser()
```

## ***Secure Connections***

Most secure connection functionality is handled outside the EJW application environment, as SSL (https) is set up at the container level, and a secure connection is then achieved simply by using “https” (HTTP over SSL) instead of “http”.

You can use EJW to restrict incoming requests via the “secureConnection” attribute in the request definition:

```
<request id="customer.ejw" secureConnection="true"  
        handlerClass="webapp.CustomerHandler" />
```

You can also define a redirect:

```
<redirect id="customerNotSecure" key="notSecureError" protocol="https"
```

```
url="customer.ejw" />
```

EJW can also be used to provide assistance with the “https”/”http” protocol in soft-coded URLs via request forwarding, as all urls, forward, and redirect definitions are also available in page authoring via:

**`${ejwUrl.Id}`**

The `${ejwUrl.id}` returns the URL defined in the url, forward, or redirect definition. The protocol attribute can be used to force “https” for secure connections. It can also be used to force the use of “http” for pages that don’t need secure connections. With this functionality you can have web applications that have secure and insecure pages intermixed and still have it all work properly.

## ***URL Rewriting***

URL rewriting is a technique that allows for browsers that have completely restricted cookies. Most session tracking is done via cookies, and if the user has cookies turned off, session tracking can’t be handled.

However, Java provides a “get-around” for this problem via URL rewriting. URL rewriting is simply tagging the URLs in a web application with a unique identifier that can be used to track the user’s session.

To the user, URL rewriting looks like this:

<http://host/servletpath;jsessionid=3F4A133A083745077...?c=intro>

in URL float-overs and URL address lines. The “jsessionid” is then used in all URLs to track the user’s session in the web application. In order for this to work, all URLs have to be rewritten using the “jsessionid”. Any non-rewritten URLs will break the web application’s session-tracking abilities.

EJW provides URL rewriting functionality via the “rewriteUrl” attribute in a forward/redirect definition. URL rewriting is provided by default (i.e. `rewriteUrl="true"`).

## **Appendix A – Validation**

**NOTE:** All validations are performed on the request parameter returned by `ServletRequest.getParameter(parameterName)`.

All arguments are values compared to the parameter value and separated by commas ‘,’.

All strings are character-case insensitive, except where an “IgnoreCase” version exists.

Empty data is not considered an error by default, so you must check specifically for an empty or null value using “isEmptyOrNull”, as other validations do not consider it an error if the value is empty, null or not found.

All validations are also available to you via code from `ServerInterface.getValidationHandler()`.

---

### *endsWith*

Checks if parameter value ends with “str”.

**Arguments**

<b>str</b>	String to compare the end with
------------	--------------------------------

### *endsWithIgnoreCase*

Checks if parameter value ends with “str”; ignores character case.

**Arguments**

<b>str</b>	String to compare the end with
------------	--------------------------------

### *isByte*

Checks if parameter is a valid byte value.

**No Arguments Needed**

### *isCreditCard*

Checks if parameter is a valid credit card number.

**No Arguments Needed**

### *isDate*

Checks if parameter validates as a date formatted with the date format argument.

**Arguments**

<b>format</b>	Date format as defined in <code>java.text.SimpleDateFormat</code>
---------------	---

*isDouble*

Checks if parameter is a valid double value.

**No Arguments Needed**

*isEmail*

Checks if parameter is a valid email address.

**No Arguments Needed**

*isEqualTo*

Checks if parameter value compares equal to "str".

**Arguments**

<b>str</b>	String to compare
------------	-------------------

*isEqualToIgnoreCase*

Checks if parameter value compares equal to "str"; ignores character case.

**Arguments**

<b>str</b>	String to compare
------------	-------------------

*isFloat*

Checks if parameter is a valid float value.

**No Arguments Needed**

## *isGreaterOrEqualToMin*

Checks if parameter value compares equal or greater to “min”.

### **Arguments**

<b>min</b>	Integer value to compare
------------	--------------------------

## *isGreaterOrEqualToMinLength*

Checks if parameter string is equal or greater to “min” length.

### **Arguments**

<b>min</b>	Integer value to compare
------------	--------------------------

## *isInRange*

Checks if parameter value is within the range of “val, val”.

### **Arguments**

<b>val, val</b>	Two integer values representing the lower/ higher ends, inclusive
-----------------	--

## *isInt*

Checks if parameter is a valid integer value.

**No Arguments Needed**

## *isIsbn*

Checks if parameter is a valid International Standard Book Number value.

**No Arguments Needed**

## *isLessOrEqualToMax*

Checks if parameter value compares equal or less to “max”.

### **Arguments**

<b>max</b>	Integer value to compare
------------	--------------------------

### *isLessOrEqualToMaxLength*

Checks if parameter string is equal or less to “max” length.

#### **Arguments**

<b>max</b>	Integer value to compare
------------	--------------------------

### *isLong*

Checks if parameter is a valid long value.

**No Arguments Needed**

### *isNotEmptyOrNull*

Checks if parameter is not empty or null.

**No Arguments Needed**

### *isEmptyOrNull*

Checks if parameter is empty or null.

**No Arguments Needed**

### *isShort*

Checks if parameter is a valid short value.

**No Arguments Needed**

### *isSubstring*

Checks if parameter is a substring of “str”.

#### **Arguments**

<b>str</b>	Check if value is a substring of “str”
------------	--

### *isSubstringIgnoreCase*

Checks if parameter is a substring of “str”; ignores character case.

**Arguments**

<b>str</b>	Check if value is a substring of "str"
------------	--

*isTrueFalse*

Checks if parameter value equals "true" or "false".

**No Arguments Needed**

*isUrl*

Checks if parameter value is formatted as a URL.

**No Arguments Needed**

*isYesNo*

Checks if parameter value equals "yes" or "no".

**No Arguments Needed**

*matchesRegExp*

Checks if parameter value compares to the regular expression.

**Arguments**

<b>str</b>	Regular expression as defined for String.matches
------------	--

*startsWith*

Checks if parameter value starts with "str".

**Arguments**

<b>str</b>	String to compare the beginning with
------------	--------------------------------------

*startsWithIgnoreCase*

Checks if parameter value starts with "str"; ignores character case.

**Arguments**

<b>str</b>	String to compare the beginning with
------------	--------------------------------------

## Appendix B – EJW DTD

```
<!ELEMENT EJW (request | url | validation | message)*>
```

```
<!ELEMENT request (parameter | security | forward | redirect | validation |  
    forwardReference | redirectReference | validationReference)*>
```

```
<!ATTLIST request  
    id ID #REQUIRED  
    match CDATA #IMPLIED  
    matchRegExp CDATA #IMPLIED  
    secureConnection (true | false) #IMPLIED  
    handlerClass CDATA #IMPLIED  
    handlerMethod CDATA #IMPLIED  
>
```

```
<!ELEMENT parameter EMPTY>
```

```
<!ATTLIST parameter  
    name CDATA #REQUIRED  
    value CDATA #REQUIRED  
>
```

```
<!ELEMENT security EMPTY>
```

```
<!ATTLIST security  
    role CDATA #REQUIRED  
>
```

```
<!ELEMENT url EMPTY>
```

```
<!ATTLIST url  
    id ID #REQUIRED  
    protocol CDATA #IMPLIED  
    url CDATA #REQUIRED  
    rewriteUrl (true | false) #IMPLIED  
>
```

```
<!ELEMENT forward EMPTY>
```

```
<!ATTLIST forward  
    id ID #REQUIRED  
    key CDATA #REQUIRED  
>
```

```
protocol CDATA #IMPLIED
url CDATA #REQUIRED
rewriteUrl (true | false) #IMPLIED
```

>

<!ELEMENT forwardReference EMPTY>

```
<!ATTLIST forwardReference
  idRef IDREF #REQUIRED
  key CDATA #REQUIRED
```

>

<!ELEMENT redirect EMPTY>

```
<!ATTLIST redirect
  id ID #REQUIRED
  key CDATA #REQUIRED
  protocol CDATA #IMPLIED
  url CDATA #REQUIRED
  rewriteUrl (true | false) #IMPLIED
```

>

<!ELEMENT redirectReference EMPTY>

```
<!ATTLIST redirectReference
  idRef IDREF #REQUIRED
  key CDATA #REQUIRED
```

>

<!ELEMENT validation EMPTY>

```
<!ATTLIST validation
  id ID #REQUIRED
  parameterName CDATA #IMPLIED
  type (endsWith |
    endsWithIgnoreCase |
    isByte |
    isCreditCard |
    isDate |
    isDouble |
    isEmail |
    isEqualTo |
    isEqualToIgnoreCase |
    isFloat |
    isGreaterOrEqualToMin |
    isGreaterOrEqualToMinLength |
```

```
isInRange |
isInt |
isIsbn |
isLessOrEqualToMax |
isLessOrEqualToMaxLength |
isLong |
isNotEmptyOrNull |
isEmptyOrNull |
isShort |
isSubstring |
isSubstringIgnoreCase |
isTrueFalse |
isUrl |
isYesNo |
matchesRegExp |
startsWith |
startsWithIgnoreCase) #REQUIRED
arguments CDATA #IMPLIED
errorMessage CDATA #REQUIRED
>
```

```
<!ELEMENT validationReference EMPTY>
```

```
<!ATTLIST validationReference
  idRef IDREF #REQUIRED
  parameterName CDATA #REQUIRED
>
```

```
<!ELEMENT message EMPTY>
```

```
<!ATTLIST message
  id ID #REQUIRED
  value CDATA #REQUIRED
>
```