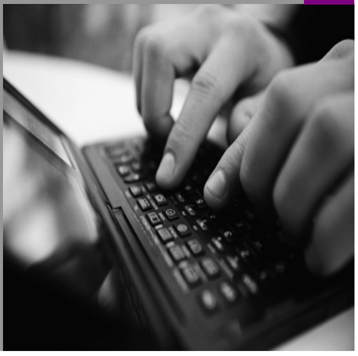


How-to Guide
SAP NetWeaver '04

How To... Configure the J2EE Engine Deployment Descriptor

Version 1.00 – July 2004

Applicable Releases:
SAP Enterprise Portal 6.0 on WEB AS 6.40

1 Introduction

This document describes how to implement the following mechanisms in SAP Enterprise Portal 6.0 on WebAS 6.40:

- Reverse Proxy Filter

  The filter enables the portal to modify HTTP responses to requests that are transmitted via a proxy server, and to respond to multiple domain names.

  For more information, see "Implementing the Reverse Proxy Filter Mechanism".

- Portal Gateway

  The gateway mechanism allows you to create aliases in the portal, and to launch for guests or anonymous users a separate Welcome page, that is different from that of named portal users.

  For more information, see "Defining the Portal Gateway Mechanism" on page 17.

The guide also provides information about the XML elements in the deployment descriptor file, *web.xml,* including the specific XML elements that implement the reverse proxy filter and the portal gateway mechanisms in the portal environment. Implementing these mechanisms allows you to modify the behavior of the portal at runtime.

2 Configuring the J2EE Deployment Descriptor (web.xml) for SAP Enterprise Portal 6.0 on WebAS 6.40

The deployment descriptor file is an XML-based text file that is deployed when you install the J2EE Engine of the SAP Web Application Server.

2.1 Accessing the Deployment Descriptor File

The file, web.xml is stored in a special directory, /WEB-INF/, which holds private files that are not accessible to clients.
You can locate the /WEB-INF/ directory in the portal installation directory:

```
<SAP SYSTEM ID>/jc_<INSTANCE_NUMBER>/j2ee/cluster/server_<INSTANCE_NUMBER>
/apps/sap.com/irj/servlet_jsp/irj/root/WEB-INF/
```

2.2 XML Elements in the Deployment Descriptor File

This section describes the XML elements that are used in the deployment descriptor. For detailed information, refer to the standard for the Java servlet API, Java™ Servlet Specification, version 2.3 at: **java.sun.com/j2ee/**.

Save a copy of the original *web.xml* file before modifying it.

When the system administrator manually removes, or installs, an instance of the J2EE Engine within the portal cluster, the initial web.xml file that was installed is automatically overwritten due to the J2EE synchronization processes.
When this happens, you lose any changes made to the contents of the web.xml file. You can duplicate the modified web.xml file and use it to replace the newly installed file on each node in the cluster. For detailed information on how to update a file in the cluster, refer to “Updating the file web.xml in the Cluster for the Portal” on page 23.

All the elements are defined within an opening and closing root element <web-app>, and </web-app>:

<display-name>	Specifies the name of the portal application to display. Use a short name that can be displayed in the Portal Platform.
<filter>	Defines a filter class and its initialization parameters.

For information on implementing the reverse proxy filter mechanism, see “Implementing the Reverse Proxy Filter Mechanism” on page 8.

The following elements can be defined in the <filter> element:

<filter-name>

Define a filter by giving it a name and specifying the filter class: the <filter-name> and <filter-class> elements.

You then define the mapping using the value of the already defined filter name.

The sub-elements in the <filter-mapping> element define when the corresponding filter will be executed.

<display-name>

A name for the filter that is displayed in the portal.

<filter-class>

Required for specifying the fully-qualified class name for the filter.

<init-param>

An optional element that contains a name/value pair. The name/value pair is used as initialization parameters for the filter.

Use a separate set of <init-param> tags for each parameter.

<filter-mapping>

Defines when to load the corresponding filter.

If you map the filter to a servlet, then the filter is executed prior to executing the corresponding servlet.

If you map the filter to a URL pattern, it is executed when the specified URL pattern is part of the request URL.

The following elements can be defined in a <filter-mapping> element:

<filter-name>

Required for defining a name for the filter to which you are mapping a servlet or URL pattern. The value defined must be the same as the value <filter-name> in the <filter> element.

<url-pattern>

Specifies the URL pattern you map to the filter.

It is the part of the URL after the scheme, hostname, port, and path, **http://host:port/path** that is compared to the defined <url-pattern> element for portal. If the patterns match, the filter mapped in this element is called.

Path is the actual context root. The URL pattern specified is relative to the application’s context root.

For example, the context root of the portal is “**irj**”. If you have a URL pattern of **/portal/help/***, then the filter is executed when the requested URL is **http://host:port/irj/portal/help/**

For example:

- /portal/help/*

- /portal/*

- /portal

- *.news

<servlet>

Specifies the name of a servlet to map to the filter.

The filter is executed prior to calling this servlet. For this reason, first define the servlet using the <servlet> sub-elements namely, <servlet-name>, <servlet-class>, and so on.

The value of this tag must be the same as the value of the <servlet-name> tag from the servlet’s definition.

<listener>

Defines the class that handles instances of the application. It has the following element:

<listener-class>

Defines the fully qualified name of the listener class that responds to an event.

<servlet >

Contains the declarative data of a servlet.

The following elements can be defined in the <servlet> element:

<servlet-name>

Define a servlet by giving it a name and specifying the servlet class: the <servlet-name> and < servlet -class> elements.

You then define the mapping using the value of the already defined servlet name.

The sub-elements in the <servlet -mapping> define when the corresponding servlet will be executed.

<display-name>

A short name that is displayed by Portal Platform.

<description>

Text description of the servlet.

<servlet-class>

Required for defining the fully-qualified class name of the servlet. Replace this element with the <jsp-file> element if you are actually declaring a JSP file.

<init-param>

Contains a name/value pair used as the initialization parameters of the servlet.

Use a separate set of <init-param> tags for each parameter.

<load-on-startup>

Required for enabling the Portal Platform to initialize the specified servlet when it starts. This element uses integers to show the order in which to load the servlets.

A servlet with a lower numeric value is loaded before a servlet with a higher value.

<servlet-mapping>	Defines mapping between a servlet and a URL pattern. The following elements can be defined in a <servlet-mapping> element: <servlet-name> Specifies the name of the servlet to which you map a URL pattern. This must be the same name as the one assigned to the <servlet-name> tag in the <servlet> declaration tag. <url-pattern> Specifies the URL pattern you map to the servlet or JSP file. It is the part of the URL after the scheme, hostname, port, and path, for instance: **http://host:port/path** that is compared to the defined <url-pattern> element for portal. If the patterns match, the servlet mapped in this element is called. *Path* is the actual context root. The URL pattern specified is relative to the application’s context root. Additional information on <url-pattern> is available in the description of <filter-mapping> discussed previously.
<welcome-file-list>	Contains an ordered list of welcome-file elements. When the URL request is a directory name, the first file specified in this element is supplied. If that file is not found, the server then tries the next file in the list. The following element can be defined: <welcome-file> An optional element that specifies the name of the file to use as a default page, such as index.html

ejb-ref	Defines a reference to an Enterprise Java Bean (EJB) resource. It is an optional element. The following elements can be defined within an <ejb-ref> element. <ejb-ref-name> The name of the EJB reference used in the portal application. <ejb-ref-type> Specifies the Java class type of the referenced EJB. <home> The fully qualified class name of the EJB home interface. <remote> The fully qualified class name of the EJB remote interface. <ejb-link> The <ejb-name> of an EJB in an encompassing J2EE application package. The value has the following format: <EJB_JAR_name>.jar#<EJB name>. <run-as> A security role whose security context is applied to the referenced EJB. Must be a security role defined with the <security-role> element.

3 Implementing the Reverse Proxy Filter Mechanism

This section describes the implementation of the reverse proxy filter mechanism in the Portal Platform.

3.1 Use Cases

The filter enables the portal to do the following:

- Modify HTTP responses to requests that are transmitted via a proxy server so that content in the portal that is referenced by absolute URLs becomes available to clients.

  For more information on the types of URL in the portal, see “URL Types” on page 9.

- Work with multiple domain names via several proxy servers.

- Use aliases for proxy servers in the portal environment.

Using the information in this section, you can manually configure the filter mechanism for proxy servers in the portal environment.
In addition, this section describes how to create your own filter mechanism for any proxy server, for example, one that is homegrown, using code samples. For additional information on code samples, see “Sample Java class” on page 16.

3.1.1 Using Third Party Mechanisms

There are other third party mechanisms, including proxy servers that can be configured to modify HTTP responses, for example, a proxy server such as, Apache 2.0.
If you are using an SSL accelerator with a proxy server, configure the filter mechanism to use the HTTPS protocol between the portal and the proxy server. You do so by specifying the value for the scheme element in the web.xml file to be HTTPS.
However, you can configure the proxy server to send a special header called *ClientProtocol*, in which case you can skip implementation of the filter mechanism.
The J2EE Engine has a property, HTTP Provider Service, with which you can change the scheme.
For detailed information about the HTTP Provider Service, go to SAP Help Portal at: **help.sap.com** → *SAP NetWeaver* → *Release '04* → *Application Platform (SAP Web Application Server)* → *Java Technology in SAP Web Application Server* → *Administration Manual* → *Server Administration* → *SAP J2EE Engine Administration* → *Web Container* →*HTTP Provider Service*.

3.2 Related Documentation

The information in this section relates only to the reverse proxy filter mechanism in SAP Enterprise Portal 6.0; it does not include discussions and recommendations for setting up a proxy server for the portal.
Check SAP note **480520** for additional information on configuring an Apache Web server for the portal environment.

3.3 What the Filter Does

The filter enables the portal to do the following:

- Distinguish between HTTP requests to the portal: those coming from external sources, such as the Internet and outside the firewall of an enterprise, and those coming from internal sources, such as the intranet within the firewall
- Replace the scheme, host, and port number in an HTTP request
- Respond to more than one domain name via several proxy servers

3.4 Distinguishing between HTTP Requests

URLs in the portal identify the location of available portal content and resources. Content can fail to display in clients due to several factors, including the following: the type of URL, and the origin of the URL.

3.4.1 URL Types

The Portal Platform contains two kinds of URL: relative and absolute URLs.

- Relative URLs

  A relative URL contains in abbreviated form the access information for an object. This allows objects in a portal to reference each other without requiring the complete reference information. In addition, the group of objects can be moved without changing any references.

  The majority of URLs in the Portal Platform are relative URLs that start with the forward slash "/" character. For example, /irj/portalapps/com.sap.portal.themes.lafservice

- Absolute URL

  An absolute URL contains the fully qualified access information for an object. This is made up of the scheme, hostname, and port number, as follows: **<scheme>://<hostname>.<domain>.<extension>:<port>/<path>/<object>**

  For example:
  `http://www.sap.com:80/irj/portalapps/com.sap.portal.themes.lafservice`

Portal content and resources that are referenced by absolute URLs are not available to clients that requested them via a proxy server. This is because the absolute URL contains the portal server's hostname, which is not visible to clients.
The filter allows the portal to alter the HTTP response so that the client can successfully display all objects via a proxy server.

3.5 Replacing Scheme, Host and Port Number

The filter mechanism modifies an HTTP response according to the source of the request.
You may want to implement other mechanisms that modify HTTP responses, instead of the filter mechanism. For information about use cases for the filter mechanism, refer to “Use Cases” on page 8.

3.5.1 Origins of HTTP requests: External and Internal Clients

The portal can identify the source of any HTTP request, either external or internal. External HTTP requests are queries coming from clients outside the DMZ, and internal HTTP requests are queries that come from the clients within the firewall.
All external HTTP requests go to the proxy server, and not directly to the portal. The proxy server obtains the responses for such requests and provides them to the client as if it were the source of the responses.
When the filter receives a request, it first determines whether the source of that request is external or internal. When the source is external, the filter provides a different implementation of the response. The regenerated implementation provides the scheme, host, and port of the reverse proxy server.
When the source is internal, the filter forwards the HTTP request without further processing. To enable the filter to distinguish between external and internal HTTP requests, do the following:

- Configure your proxy server to add HTTP header data to every request
- Configure the filter to obtain the HTTP header data in a request.

Alternatively, first define and use an alias for the proxy server, and then specify this alias in the portal gateway mechanism. For information about the portal gateway, refer to “Defining the Portal Gateway Mechanism” on page 17.
The HTTP header data consists of a header name, and a header value. For example, the header name can be “Via,” and a header value can be a name that identifies a specific proxy server.
You can implement different header data for several proxy servers in the portal landscape, and configure the filter to identify the different header data in all requests from the proxy servers.

3.6 Defining the Reverse Proxy Filter in the Deployment Descriptor

You define the reverse proxy filter as part of the portal, using the deployment descriptor, web.xml. In the deployment descriptor, declare the filter and then map the filter to a specific servlet installed by the portal. If you are working with more than one proxy server, then declare a <filter> element for each proxy server. For additional information, see “Using Multiple Domain Names to Access the Portal
” on page 13.

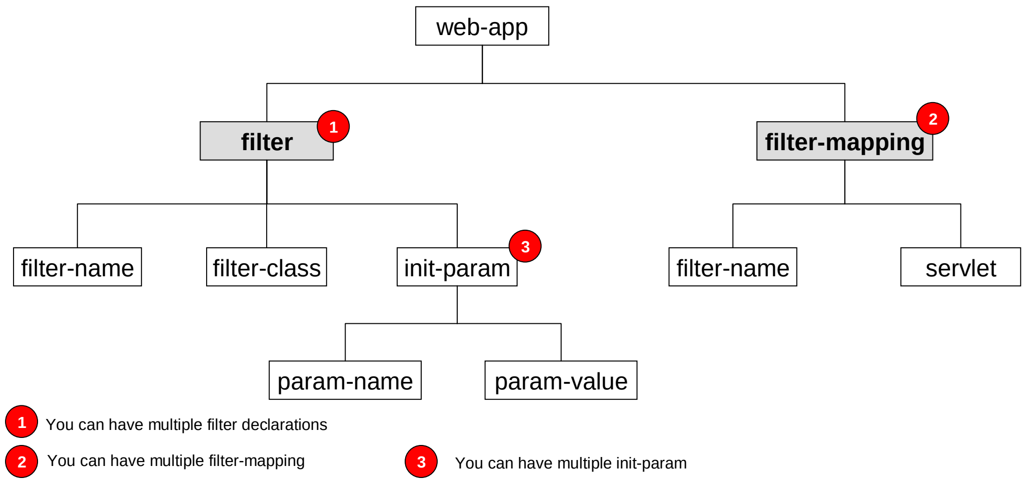
3.6.1 Requirements

You can activate the reverse proxy filter under the following conditions:

- You have deployed SAP Enterprise Portal 6.0 SP2 or higher in a secure network environment.
- You have successfully deployed and configured a proxy server that serves portal content to clients.

  If your proxy server has been configured to work with a mechanism that accelerates the authentication processes, such as an SSL accelerator, then configure the mechanism to add an extra header. For example, <ClientProtocol> = HTTPS
  If the authentication processing mechanism, such as an SSL accelerator is not configured, then you must define the HTTPS value in the relevant XML code for the reverse proxy filter, so that the response scheme is always HTTPS.

The following diagram illustrates the elements used for the reverse proxy filter mechanism:


web-app
filter
1
filter-mapping
2
filter-name
filter-class
init-param
3
filter-name
servlet
param-name
param-value
1 You can have multiple filter declarations
2 You can have multiple filter-mapping
3 You can have multiple init-param

To define the filter:

1. Open the web.xml file in a text editor.
2. Add the filter declaration. The <filter> element must precede the <listener> and <servlet> elements.
   The <filter> element declares the filter, defines a name for the filter, and specifies the Java class that executes the filter.
   For example:

```
<filter>
   <filter-name> ReverseProxyFilter </filter-name>
   <filter-class>
        com.sapportals.portal.crosstopics.reverseproxyfilter.ReverseProxyFilter
    </filter-class>
    <load-on-startup> 1 </load-on-startup>
    <!-- Limits the requests to a specified scheme, such as HTTP, HTTPS. When empty, the <param-
    value> is taken from the request. -->
    <init-param>
        <param-name> scheme </param-name>
        <param-value> … </param-value>
```

```
</init-param>
<!-- Specifies the name of the proxy server machine and domain name.-->
<init-param>
<param-name> proxy-host-name </param-name>
<param-value> lego.tlvp.sap.corp </param-value>
</init-param>
<! -- Port for HTTP requests: specifies the port number of the proxy Web server. -->
<init-param>
    <param-name> proxy-port-http </param-name>
    <param-value> 80 </param-value>
</init-param>
<!— Port for HTTPS requests: it specifies the port number of the secure proxy Web server. -->
<init-param>
    <param-name> proxy-port-https </param-name>
    <param-value> 443 </param-value>
</init-param>
<!-- The <param-value> for the “filter-header-name” is the HTTP header to search for in the
request. -->
<!— If you have defined an alias for the proxy server, the filter takes the alias and ignores the
“filter-header-name” and the “filter-header-value”. -->
<init-param>
    <param-name> filter-header-name </param-name>
    <param-value> Via </param-value>
</init-param>
<!— The value to look for in the header. It specifies the HTTP header value. Note that if not defined,
then only the filter-header-name in the request is used to determine the external source. -->
<init-param>
    <param-name> filter-header-value </param-name>
    <param-value> lego.tlvp.sap.corp </param-value>
</init-param>
<! – Activates the debugging mode for the filter. It takes true or false. The default value is false.
True enables the filter to print debugging messages to the console. False disables printing
debugging messages. -->
<init-param>
    <param-name> debug </param-name>
    <param-value> false </param-value>
</init-param>
</filter>
```

Remove the comments if you copy and paste the text in this example.

3. Add filter mappings. Define the <filter-mapping> element for a particular servlet, and then map the filter to the name of that servlet.
   The <filter-mapping> element specifies which filter to execute based on the servlet name. It must immediately follow the <filter> element.
   For example, the following maps the filter, *ReverseProxyFilter* to the servlet, *com.sapportals.portal.crosstopics.reverseproxyfilter.ReverseProxyFilter*:

```
<filter-mapping>
      <filter-name> ReverseProxyFilter </filter-name>
      <servlet-name> com.sapportals.portal.crosstopics.reverseproxyfilter.ReverseProxyFilter </servlet-
    name>
</filter-mapping>
<filter-mapping>
    <! –  The <filter-mapping> element defines the servlet for the filter. This bonds the filter to any
    activated PRT servlet. Make sure that you define the same <filter-name> value as specified in the
    <filter> element above. -->
    <filter-name> ReverseProxyFilter </filter-name>
    <servlet-name> prt </servlet-name>
</filter-mapping>
<filter-mapping>
    <! –  Captures the default servlet activator. In a multiple proxy server environment, make sure that
    the <filter-name> value is the same as the value defined in the <filter-name> tag at the beginning of
    the code. -->
    <filter-name> ReverseProxyFilter </filter-name>
    <url-pattern> /servlet/* </url-pattern>
</filter-mapping>
<filter-mapping>
    <!– Captures all activated JSP files. In a multiple proxy server environment, make sure that the
    <filter-name> value is the same as the value defined in the <filter-name> tag at the beginning of the
    code. -->
    <filter-name> ReverseProxyFilter </filter-name>
    <url-pattern> *.jsp </url-pattern>
</filter-mapping>
```

3.7 Using Multiple Domain Names to Access the Portal

You can enable access to the portal via multiple domain names. To access the portal through several different domain names, you must have several proxy servers, each with a different domain name. You then activate the filter mechanism and configure it to work with several proxy servers.
Repeat both the <filter> and <filter-mapping> declarations for each server, when working with several proxy servers. Afterwards, modify the values for the repeated elements to make the portal aware of each proxy server.

Make sure that the declared <filter> element for each proxy server does the following:

- Specifies the Java class file

  The deployment descriptor for the portal uses the same Java class file to activate the filter mechanism for each proxy server.

- Defines different <filter-name> values for use with each proxy server

  For each proxy server, declare the same <filter-name> value in the <filter-mapping> element as that declared in the <filter> declaration.

- Defines different “filter-header-name” and its corresponding <param-value> value for use with each proxy server.

For each <filter> element declared in the deployment descriptor for a proxy server, define both the name of the specific proxy server machine and its domain name.

Example

This scenario consists of two proxy servers with different domain names in the portal environment.

The following describes the deployment descriptor (web.xml) for the first proxy server with the name, **lego-sap.com:**

```
<filter>
   <filter-name> LegoProxyFilter </filter-name>
   <filter-class>
        com.sapportals.portal.crosstopics.reverseproxyfilter.ReverseProxyFilter
    </filter-class>
    <load-on-startup> 1 </load-on-startup>
    <init-param>
        <param-name> scheme </param-name>
        <param-value>  </param-value>
    </init-param>
    <init-param>
        <param-name> proxy-host-name </param-name>
        <param-value> lego-sap.com </param-value>
    </init-param>
    <init-param>
        <param-name> proxy-port-http </param-name>
        <param-value> 80 </param-value>
    </init-param>
    <init-param>
        <param-name> proxy-port-https </param-name>
        <param-value> 443 </param-value>
    </init-param>
    <init-param>
        <param-name> filter-header-name </param-name>
        <param-value> Via </param-value>
    </init-param>
    <init-param>
        <param-name> filter-header-value </param-name>
        <param-value> lego-sap.com </param-value>
    </init-param>
    <init-param>
        <param-name> debug </param-name>
    <param-value> false </param-value>
    </init-param>
</filter>
<filter-mapping>
  <filter-name> LegoProxyFilter </filter-name>
```

```
            <servlet-name>
                    com.sapportals.portal.crosstopics.reverseproxyfilter.Reverse
                    ProxyFilter
            </servlet-name>
    </filter-mapping>
```
To define the settings for the proxy server, waldorf.tv.com, declare new <filter> and
<filter-mapping> elements in the deployment descriptor (web.xml):
```
      <filter>
          <filter-name> WaldorfProxyFilter </filter-name>
          <filter-class>
                com.sapportals.portal.crosstopics.reverseproxyfilter.ReverseProxyFilter
           </filter-class>
           <load-on-startup> 1 </load-on-startup>
           <init-param>
                <param-name> scheme </param-name>
                <param-value>  </param-value>
           </init-param>
           <init-param>
                <param-name> proxy-host-name </param-name>
                <param-value> waldorf.tv.com </param-value>
           </init-param>
           <init-param>
                <param-name> proxy-port-http </param-name>
                <param-value> 80 </param-value>
           </init-param>
           <init-param>
                <param-name> proxy-port-https </param-name>
                <param-value> 443 </param-value>
           </init-param>
           <init-param>
                <param-name> filter-header-name </param-name>
                <param-value> Via </param-value>
           </init-param>
           <init-param>
                <param-name> filter-header-value </param-name>
                <param-value> waldorf.tv.com </param-value>
           </init-param>
           <init-param>
                <param-name> debug </param-name>
                <param-value> false </param-value>
           </init-param>
    </filter>
    <filter-mapping>
        <filter-name> WaldorfProxyFilter </filter-name>
            <servlet-name>
                    com.sapportals.portal.crosstopics.reverseproxyfilter.Reverse
                    ProxyFilter
            </servlet-name>
    </filter-mapping>
```
With these definitions, the portal has been enabled to work with two proxy servers and two domain names. The portal can provide responses to client requests in the form of the following URLs:

- http://lego-sap.com/, and https://lego-sap.com
- http://waldorf.tv.com, and https://waldorf.tv.com

3.8 Creating a Reverse Proxy Filter

You can create a reverse proxy filter mechanism to use with your homegrown proxy server. If you choose to create and implement your own filter solution for the portal environment, do the following:

1. Write a Java class for implementing the filter.
2. Implement your own request and response wrapper mechanisms.
3. Configure the settings for your implementation in the file, *web.xml*.

Create a JAR file that consists of your Java class, the request and response wrapper mechanisms, and then add the new JAR file to the folder: …\cluster\server<instance_number>\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\lib

3.8.1 Sample Java class

Here is sample Java code for developing your own reverse proxy filter solution:

```
public class reverseproxysupport implements Filter
{
            public void init(FilterConfig arg0) throws ServletException{}
            public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException
{
   ServletRequest sr = new RequeustWrapper((HttpServletRequest)request);
   ServletResponse sResponse = new ResponseWrapper((HttpServletRequest)sr,
   (HttpServletResponse)response);
   chain.doFilter(sr, sResponse);
}
public void destroy() {}
}
```

3.8.1.1 RequestWrapper and ResponseWrapper Implementation

Here is sample Java code for developing your own RequestWrapper and ResponseWrapper mechanisms:

```
public class RequestWrapper extends HttpServletRequestWrapper
{
   public String getServerName() {…}
   public int getServerPort() {…}
   public String getScheme() {…}
}
public class ResponseWrapper extends HttpServletResponseWrapper
{
   public void sendRedirect(String redirectURL) throws IOException {…}
}
```

3.8.1.2 Configuring the Portal for Your Reverse Proxy Filter Solution

You configure the portal to work with your filter solution by adding the relevant XML elements to the file, *web.xml*.

For additional information on reverse proxy filter definitions in the portal deployment descriptor, web.xml, refer to “Defining the Reverse Proxy Filter in the Deployment Descriptor” on page 10.

4 Defining the Portal Gateway Mechanism

The portal gateway mechanism serves as an entrance to the portal. It allows you to modify the behavior of the portal by:

- Customizing URLs to portal applications
- Using aliases for custom URLs
- Defining suitable portal pages for users based on their authorization level and authentication schemes

This section describes the following:

- How to assign names (aliases) to custom URLs in the portal to enable easy access to portal content via different URLs.

  For example, having defined more than one entry URL, you can configure the gateway to direct a URL to a high or low speed connection.
- How to enable the portal to launch an initial page based on users' authentication

4.1 The Gateway Mechanism in the Portal

The SAP Web Application Server interfaces with the gateway mechanism to enable the portal to identify the custom URLs defined, and the names that you assign to these URLs in the deployment descriptor.
In an out-of-the-box portal, there are three predefined aliases to the following customized URLs in the portal:

- /portal

  You can specify this name after the path, /irj/, to launch the portal. Use it to access directly the assigned initial portal page and desktop. For example: **http://sapinside.com:50100/irj/portal**
- /portal/light

  Specify this alias after the path /irj/ in the portal URL to directly access the assigned initial portal page and desktop over low bandwidth connection.
  For example: **http://sapinside.com:50100/irj/portal/light**
- /portal/anonymous

  Specify this alias after the path /irj/ in the portal URL to directly access the assigned initial portal page and desktop for users that are authenticated as guests.
  For example: **http://sapinside.com:50100/irj/portal/anonymous**

You can define specific initial portal pages and desktops for different users based on their authentication schemes, using the portal display rules at runtime. Before you define a display rule using a portal URL alias, configure your gateway mechanism accordingly.

For more information on using display rules, go to SAP Help Portal at: **help.sap.com/nw04** → *SAP NetWeaver → Release '0 → English → SAP NetWeaver → People Integration → Portal → Administration Guide → System Administration → Portal Display → Portal Display Rules* (*Rule Collections*) → *Portal URL Aliases*

You can add several aliases for various contents in the portal via the gateway mechanism. Each alias has its own configuration settings and must contain the following:

- The name to use as an alias. For example: /sapnet
- The attribute of the alias which defines whether it permits guests or named users in the portal
- The attribute of the alias which defines whether it can be used via a proxy server or directly in the URL to launch the portal.
- The attribute of the alias which defines whether it is used for low or high bandwidth connections
- The attribute of the alias which defines whether the default path of the portal runtime, "/irj/" and the alias are provided as part of the URL information returned to the client.

Each time you launch the portal with a URL that contains an alias, the gateway is activated. The gateway mechanism obtains the details of the alias, and stores them with the session of a cookie. As part of the cookie session, the data for the alias is always available to all portal components, services and other servlets. When the session is destroyed, the cookie is also destroyed.

The XML elements that define the gateway configuration consist of two parts: the gateway servlet declaration <servlet> element, and the gateway servlet mapping declaration <servlet-mapping>

Gateway Servlet Declaration

The <servlet> element for the gateway declares the following:

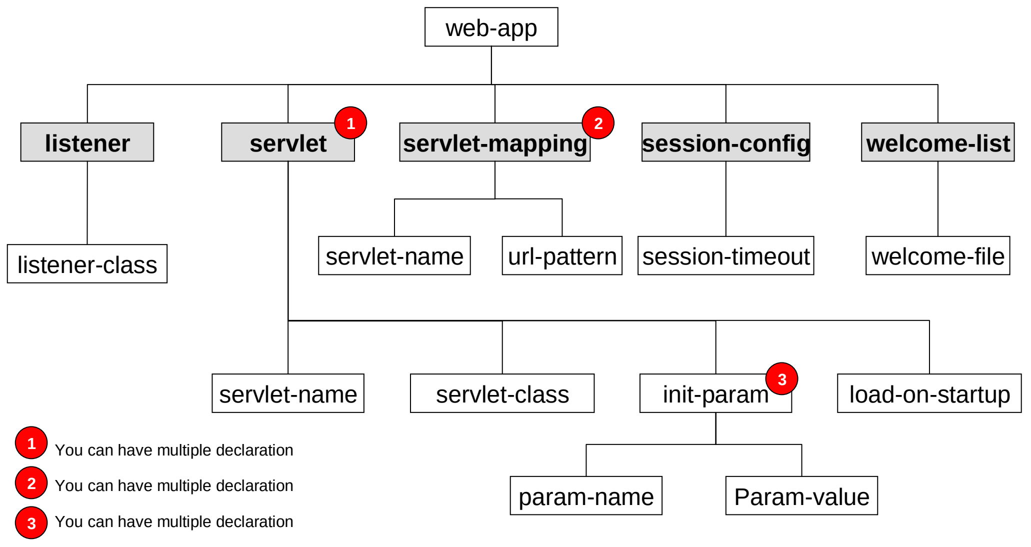
- Name of the portal gateway mechanism
- Java class of the servlet that implements the gateway mechanism
- URL to the initial portal page
- URL to an anonymous portal page
- Assigned names to URLs

Gateway Servlet Mapping Declaration

The <servlet-mapping> element configures the SAP Web Application Server to call the gateway servlet, which implements the default operations such as aliases, and to provide the specified URL (page) based on the user's authentication.

For more information on implementing the <servlet> and the <servlet-mapping> elements, refer to "XML Elements in the Deployment Descriptor" on page 2.

The following illustrates the XML elements for implementing the gateway mechanism:



4.2 Customizing URLs for Portal Applications

At design time, it is common to reference portal applications via URLs to the Java class or JSPs, however, it is convenient and easy to use a URL without the Java class, or JSP entry.
To customize the URL so that it shows without the Java class and JSP entry:

1. Define a name for the Java class or JSP in the <servlet name> sub element of the <servlet> element. For more information see “XML Elements in the Deployment Descriptor” on page 2.
2. Use the <servlet-mapping> element along with its <servlet-name> and <url-pattern> sub elements.
   - The <servlet-name> element defines the defined name to the Java class, or JSP.
   - The <url-pattern> element describes the relative URL to the Java class, or JSP. For example, the root of the portal runtime application is /irj/.

   Repeat the <servlet-mapping> element for each name you assign to a Java class, or JSP.

4.3 Assigning Aliases to Customized URLs

You can assign a name to a custom URL that points to a portal application (Java class and JSPs) using XML elements in the deployment descriptor, web.xml file. The name you assign represents the name defined for the Java class or JSP page referenced in the URL, which is a request to retrieve specific content. This alias can be used instead of the defined name.
You can assign several aliases which can equally invoke the functionality in the Java class without using the class name. Each alias can have attributes that modify the behavior of the portal.
For an example:

The URL, http://<hostname>:<port>/irj/**portal**, where the alias **portal**, can be used instead of http://<hostname>:<port>/irj/**prt/portal/prtroot/com.sap.portal.navigation.portallauncher.default** to launch the portal.
To define an alias:

1. Open the file, web.xml, in a text editor.
2. Insert an <init-param> element within the existing defined <servlet> element for each alias.
   The <servlet> element with its <servlet name>, and <servlet-class> sub elements allows you to assign a name to the Java class that implements the gateway mechanism. For detailed information, see XML Elements in the Deployment Descriptor on page 2.

   The <param-name> element inside the <init-param> element defines the actual pseudonym or alias, and the <param-value> element specifies the attributes applicable to that alias.
   The following is an example using the XML elements that implement the default aliases used in an out-of-the-box portal:

   Do not modify the defined default values in the web.xml file. Remove all comments that you paste into your web.xml file, if you copy this example.

```
<servlet>
        <!-- This is the servlet definition for the gateway mechanism. -->
        <servlet-name> gateway </servlet-name>
        <servlet-class> com.sap.portal.navigation.Gateway </servlet-class>
<load-on-startup> 0 </load-on-startup>
<init-param>
        <!— Specifies the default portal page to launch.  -->
        <param-name> portal_entry_point </param-name>
        <param-value> /servlet/prt/portal/prtroot/com.sap.portal.navigation.portallauncher.default </param-
        value>
</init-param>
<init-param>
        <!— Specifies the portal page to launch for guests and unnamed users in the portal.  -->
        <param-name> portal_anonymous_entry_point </param-name>
        <param-value> servlet/prt/portal/prtroot/com.sap.portal.navigation.portallauncher.anonymous
        </param-value>
</init-param>
```

```
<init-param>
        <!— Specifies the default alias (/portal) that can be used after the path /irj/.  -->
        <param-name> portal </param-name>
        <param-value>
        <!—     Use all or some of the following possible options that modify the implementation of the
        alias:

             –      anonymous
                    Determines whether guests and unnamed users can launch the initial portal page using
                    this alias. The default value for this option is 0, so that the mechanism cannot allow
                    anonymous users to log on to the portal using this alias.

             –      proxy
                    Allows or denies the use of this alias via a proxy server. The default value for this option is
                    0, so that the mechanism cannot allow requests via a proxy server to the portal using this
                    alias.

             –      low_bandwidth
                    Allows or denies the use of this alias over low bandwidth connections. The default value
                    for this option is 0, so that the mechanism cannot allow logon to the portal using this alias
                    over low bandwidth connection.

             –      include_in_url
                    Determines whether this alias shows as part of the URL returned to the client. The default
                    value for this option is 1, so that the portal can display the response URL using this alias.

             –      include_application_name_in_url
                    Determines whether the portal application name shows with this alias as part of the URL
                    returned to the client.  The default value for this option is 1, so that the response URL
                    displays the path /irj/ with this alias.   -->
        anonymous=0,proxy=0,low_bandwidth=0,include_in_url=1,include_application_name_in_url=1
        </param-value>
</init-param>
<init-param>
        <!— Specifies the default alias (/portal/light ) used for low bandwidth connection. See above for
              information on how to apply the various options in the <param-value> element  -->
        <param-name> portal/light </param-name>
        <param-value>
        anonymous=0,proxy=0,low_bandwidth=1,include_in_url=1,include_application_name_in_url=1
        </param-value>
</init-param>
<init-param>
        <!— Specifies the default alias (/portal/anonymous) defined for users that are authenticated as
              guests. See above for information on how to apply the various options -->
        <param-name> portal/anonymous </param-name>
        <param-value>
     <! -- See above for information on how to apply the various options in the <param-value> element  --
     >
     anonymous=1,proxy=0,low_bandwidth=0,include_in_url=1,include_application_name_in_url=1
     </param-value>
</init-param>
</servlet>
```

3. In the <servlet-mapping> declaration, insert the gateway, and then map it to the name of the servlet.
   The <servlet-mapping> element specifies which servlet to execute based on the servlet name. It must immediately follow the <servlet> element.
   For example, the following maps the gateway definition to the servlet:

```
<servlet-mapping>
        <!— Use the same <servlet-name> in the <servlet-mapping> element as defined in the <servlet>
              element for the gateway mechanism. See step 2. -->
        <servlet-name> gateway </servlet-name>
        <url-pattern>
        <!-- The <url-pattern> element defines the configuration for recognizing all names that have been
              assigned for use with the alias, "/portal".
              When the portal is configured to use multiple URL aliases, make sure that you use the correct
              URL syntax to launch the portal.
              For instance, instead of "http://<server>:<port>/irj"  use "http<server>:<port>/irj/<gateway
              mapping>" where <gateway mapping> is the string matching each <servlet-mapping>
              specified in the deployment descriptor, web.xml.
              For example, based on the default configuration the following are valid URLs to the portal:

              –     http://<server>:<port>/irj/portal

              –     http://<server>:<port>/irj/portal/light -  for example, for low bandwidth users

              –     http://<server>:<port>/irj/portal/anonymous
              The following is not a valid URL to the portal as the alias "guest" is not defined in the <servlet>
              declaration (see step 2 ), even though the url-pattern "/portal/*" is declared in the <servlet-
              mapping> element:

              –     http://<server>:<port>/irj/portal/guest
         -->
              /portal/*
        </url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name> prt </servlet-name>
        <url-pattern>
        <!-- The <url-pattern> element that defines the configuration for all aliases specified in the portal. -->
               /irj/*
        </url-pattern>
</servlet-mapping>
```

4.4 Provide Different Points of Entry into the Portal

By default, the gateway mechanism enables the portal to launch two different pages for users: a page for named portal users and a separate page for anonymous users. Similarly, the portal can be enabled to detect a high or low bandwidth connection, and direct users to a page that suits their connection.
The mechanism always needs to know which initial page to invoke for a user. However, after the initial logon, a user can change the logon action, for example, from anonymous mode to a named user mode.
The following is an example of the default declaration for launching separate pages for both named and anonymous users.

⚠

Do not modify the defined default values.

```
<init-param>
    <param-name> portal_entry_point </param-name>
    <param-value> /servlet/prt/portal/prtroot/com.sap.portal.navigation.portallauncher.default
    </param-value>
</init-param>
<init-param>
    <param-name> portal_anonymous_entry_point </param-name>
    <param-value> /servlet/prt/portal/prtroot/com.sap.portal.navigation.portallauncher.anonymous
    </param-value>
</init-param>
```

5 Updating the file web.xml in the Cluster for the Portal

You must configure the cluster environment after modifying the file, web.xml. Doing so updates the cluster about any changes to the portal.
Configure the cluster for the portal using Single File Update in the J2EE Engine Visual Administrator tool. More than one file in the cluster is simultaneously updated using Single File Update.

To update the cluster about the modified web.xml file:

1. Open the Visual Administrator, on the machine with the Central Instance for the J2EE Engine.
    - Select the go (go.bat on Windows) file in the portal installation path: …/usr/sap/<system id>/jc<number>/j2ee/admin/
2. From the administration console, select **Services** →**Deploy**
3. Select **Single File Update** in the right side of the console, and specify the following:

    Application = sap.com/irj

    Module = epbc.war

    File name = <Select the file you want to upload to the database for the J2EE Engine>

    Mapping = WEB-INF/deployment/< file name>

    Container = servlet_jsp

    For example, if the file to be updated is web.xml, then the mapping is WEB-INF/web.xml. If for instance, the file is com.sap.portal.usermanagement.par,

then the mapping is
WEB-INF/deployment/com.sap.portal.usermanagement.par

4. Select **Add**, and then **OK**.

   From the messages at the bottom of the window, verify the details for the file to upload.

5. Restart the J2EE Engine.