|  |
| --- |
| **R01HP AppEmbed Filter** |
| Content  [1 R01HP: What is Portal Application embedding 2](#_Toc520886253)  [2 Inside view: how does it work 4](#_Toc520886254)  [2.1 Running R01HP as a Java Web App servlet filter 4](#_Toc520886255)  [2.2 Running R01HP as a stand-alone reverse proxy 6](#_Toc520886256)  [3 System operation 8](#_Toc520886257)  [3.1 Running R01HP as a Java Web App servlet filter 8](#_Toc520886258)  [3.1.1. Config 8](#_Toc520886259)  [3.2 Customized R01HP AppEmbed Filter 12](#_Toc520886260)  [3.3 Running R01HP as a stand-alone reverse proxy 13](#_Toc520886261)  [3.3.1. Config 13](#_Toc520886262)  [3.3.2. A note for configuring applications running in IIS 15](#_Toc520886263)  [3.3.3. Testing the configuration 16](#_Toc520886264)  [3.4 Metrics 17](#_Toc520886265)  [4 Developer useful info 18](#_Toc520886266)  [4.1.1. General rules 18](#_Toc520886267)  [4.1.2. [R01HP] Filter 20](#_Toc520886268)  [4.1.3. Setting a non en-bedding behavior 20](#_Toc520886269)  [4.1.4. Legacy [R01H] 21](#_Toc520886270) |

1. R01HP: What is Portal Application embedding

Using [Portal Application Embedding], any [application] –*no matter which technology is it base on*- can be shown inside a [portal page] made using [PLATEA Portal Manager]:



The [application] can be *embedded* (shown) in any [portal page]: the [portal page] will decorate the embedded [application] with the site elements such as menus, banners, footers, etc.

As such, an [application] {app1} can be displayed in the portal {portalX} at the page {pageX1}:

http://site/{portal**X**}-{page**X**1}/{lang}/{app**1**}

… if required, the same application can also be accesed within another portal {portalY} and page {pageY1} (*that surely will have a different visual structure and [Visual Areas] disposition*):

http://site/{portal**Y**}-{page**Y**1}/{lang}/{app**1**}

… in the same way, the same [container portal pages] can hold another [application] {app2}:

http://site/{portal**X**}-{page**X**1}/{lang}/{app**2**}

http://site/{portal**Y**}-{page**Y**1}/{lang}/{app**2**}

Think of it as a kind of “*IFRAME”* inside the [portal page] that can *embed* (show) any [application] (*or any url in general if it can be reached from the portal app server where [R01HP] resides*)

It’s a “kind-of” IFRAME **but it’s NOT an IFRAME** (*IFRAME use is discouraged from the accessibility point of view*): from the web browser point of view the [portal page] with the embedded [application] is seen as a single whole web page but down the stream there are two **independent** components:

* The [portal page] that contains (*embeds*) the application: the [container page]
* The [application] itself

Simple enough! But things are not so easy: the **returned assembled HTML must be well formed** and this is where things turn tricky:

* The [portal container page] is a well-formed HTML page with it’s <html>, <head> and <body> sections

|  |
| --- |
| <html>  <head>{portal page head section}</head>  <body>  <div id=’container’>  {\_\_\_this is where the app HTML will be embedded\_\_\_}  </div>  </body>  </html> |

* The [application] usually is also a well-formed HTML page, also with it’s <html>, <head> and <body> sections

|  |
| --- |
| <html>  <head>{app head section}</head>  <body>  {APP HTML BODY MARKUP}  </body>  </html> |

when embedding the app, the HTML markup of [portal page] and HTML markup of the [app] **must be mixed** to create a well-formed HTML markup.

If nothing is done when wrapping the [application] HTML with it’s <html>, <head> and <body> sections inside a [container portal page] with it’s own <html>, <head> and <body> sections, **the resulting HTML wont be well formed**:

|  |
| --- |
| <html>  <head>  {portal page head section}  </head>  <body>  <div id=’container’>  <html>  <head>{app head section}</head>  <body>  {APP HTML BODY MARKUP}  </body>  </html>  </div>  </body>  </html> |

… so while embedding the [application] HTML into the [container portal page], the [application] HTML must be parsed to dissasemble the <html>, <head> and <body> and embed each section in the corresponding section of the [container portal page]:

|  |
| --- |
| <html>  <head>  {app head section}  {portal page head section}  </head>  <body>  <div id=’container’>  {APP HTML BODY MARKUP}  </div>  </body>  </html> |

**COMPATIBILITY NOTE:**

The previous versión of [R01H] was NOT capable of “reorder” the [application]’s HTML <html>, <head> and <body> sections so **the [application] must return HTML content without** **<html>, <head> and <body> sections: it must return just the [application] HTML wrapped within two** <div> **tags.**

In order to set the <head> content, a complicated, error-prone and un-maintainable apache config must be done.

… **this is no longer true**: the new [R01HP] does NOT imposes any restriction to the [application] HTML so it can be an arbitrary HTML.

1. Inside view: how does it work

Technically [R01HP] can be run in two very diferent ways:

|  |  |  |
| --- | --- | --- |
|  | How it’s deployed | When to use |
| As an stand-alone [reverse proxy] | An stand-alone reverse proxy is deployed independently of the target web app  The reverse proxy can be run in a dedicated java web app server or if the target app also runs inside a java app server, it can be deployed as an independent WAR inside the same app server as the target app | Apps in any technology (*java web app, .NET, PHP, etc*)  Java Web Apps that cannot be modified to add a servlet filter |
| A a Java Web App [servlet filter] | As a [servlet-filter] for the java web app | Java Web Apps that can be modified to include a servlet filter (*by modifying the web.xml file*) |

* 1. Running R01HP as a Java Web App servlet filter

[R01HP] is deployed as a [servlet filter] that:

1. Loads the [portal container page] that has a designated area (*a [Visual Area]*) to hold any [content] or [applications] (*[Container Visual Area]*)
2. Lets the filter run the [app] code and generate the html content
3. Joins / assembles / embed the [application] html content inside the [Visual Area] designated to hold [content] or [applications]

As seen below the flow is:

|  |  |
| --- | --- |
|  | 1. The [web server] (*apache*) detects the URL as a [portal page] embeded application if it matches a pattern like:   http://site/{portal}-{page}/{lang}/{appURL}   1. The [request] is routed through a proxy to the [target application server] where the target [java web app] resides 2. [R01HP] filter, using the {portal} and {page} info, [R01HP] loads a copy of the [container portal page] and parses it to get the <html>, <head> and <body> sections   [R01HP] filter just let the filter target servlet run and gets the generated HTML, parses it to get the <html>, <head> and <body> sections   1. The whole HTML to be returned is assembled merging the [container portal page]’s and [application]’s <html>, <head> and <body> sections 2. The [web server] (*apache*) solves other static [Visual Areas] and returns the complete HTML |

A more detailed view where the different urls and request parameters can be seen here:



* 1. Running R01HP as a stand-alone reverse proxy

[R01HP] is deployed at a [portal application server] and acts as a **reverse proxy** that:

1. Loads the [portal container page] that has a designated area (*a [Visual Area]*) to hold any [content] or [applications] (*[Container Visual Area]*)
2. Uses a reverse proxy to reach the [application server] where the [application] resides and loads it’s html content
3. Joins / assembles / embed the [application] html content inside the [Visual Area] designated to hold [content] or [applications]

As seen below the flow is:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1. The [web server] (*apache*) detects the URL as a [portal page] embeded application if it matches a pattern like:   http://site/{portal}-{page}/{lang}/{appURL}   1. The [request] is routed through a proxy to the [portal application server] where [R01HP] resides   A [web server] (*apache*) rewrite map is used to know what [portal application server (R01HP)] will handle the [application] integration.  The map is something like:   |  |  | | --- | --- | | App1 | Portal app server A | | App2 | Portal app server B | | App 3 | Portal app server C | | App 4 | Portal app server A |  1. At the [portal app server], using the {portal} and {page} info, [R01HP] loads a copy of the [container portal page] and parses it to get the <html>, <head> and <body> sections 2. Using another proxy to the [application server] where the [application] resides, [R01HP] retrieves the [application HTML] and parses it to get the <html>, <head> and <body> sections   The [portal app server] has a proxy configuration for every target [app server]   1. The whole HTML to be returned is assembled merging the [container portal page]’s and [application]’s <html>, <head> and <body> sections 2. The [web server] (*apache*) solves other static [Visual Areas] and returns the complete HTML |

A more detailed view where the different urls and request parameters can be seen here:



As seen, there’re two proxies in place that offers a lot of elasticity to the system:

|  |  |
| --- | --- |
| A proxy from the [web server] to the [portal application server] | There can be as many [portal applicaton servers] (*that holds [R01HP]*) as needed  Usually just two nodes for [portal application server] is more than enought |
| A proxy from the [portal application server] to the targer [application server] | [R01HP] has a kind of “pipes” (*proxies*)to target [application servers] where many [applications] can reside so there’s no need to configure a “pipe” (*a proxy*) for every target [application]: usually the proxy is configured for the target [application server] as a whole. |

This two-layer of proxies provides with a lot of flexibility; two of the most common configuration examples could be:

|  |  |
| --- | --- |
| Segmentation using a **single [portal application server]** |  |
| Segmentation using **multiple [portal application server]**s to multiple target [application server]s |  |

1. System operation
   1. Running R01HP as a Java Web App servlet filter
      1. Config

The use of [R01HP app embed servlet filter] in a java web app cannot be easer, just **setup the filter in the web.xml config file:**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*   id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>r01hpTestWar</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>    <!--  R01HP in FILTER MODE (does NOT uses a proxy)  -->  <filter>  <filter-name>portalPageAppEmbedServletFilter</filter-name>  <filter-class>r01hp.portal.appembed.R01HPortalPageAppEmbedServletFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>portalPageAppEmbedServletFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>    <servlet>  <description></description>  <display-name>my servlet</display-name>  <servlet-name>MyServelt</servlet-name>  <servlet-class>com.company.MyServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>MyServlet</servlet-name>  <url-pattern>/MyServlet/\*</url-pattern>  </servlet-mapping>  </web-app> |

By default [R01HP app embed filter] tries to use the config at /config/r01hp/r01hp.appembed.properties.xml

(*if this file is NOT found, a default property set is used*)

Either if this file is NOT present or custom properties are needed, **some of the config properties can be overridden at the** **web.xml** file using the servlet filter’s init params

The r01hp.appembed.properties.xml file has the following properties:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <portalpageappembedfilter environment =*"local"*>  <!-- Resources NOT embedded into a portal page ============================================ -->  <!-- A list of regular expressions that will be matched agains the URL path of the resource -->  <notEmbeddedResources>  <!--  <urlPathRegExp>/not-embeded/.\*</urlPathRegExp>  -->  </notEmbeddedResources>    <!-- Portal server configuration ==================================================== -->  <!-- Defines the location (filesystem path) of the container pages where -->  <!-- the app will be embedded, the default page to use and how these pages are cached -->  <!-- Multiple environments can be configured in the same file -->  <portalServer>  <cacheConfig>  <initialCapacity>10</initialCapacity>  <maxSize>100</maxSize>  <checkInterval>20s</checkInterval>  </cacheConfig>  <portalFiles>  <!-- the filesystem path where the container pages can be found -->  <root>d:/temp\_dev/r01hp/</root>  <pages>/html/pages/portal</pages>  <!-- the default portal/page/lang to be used if none can be guess from the request -->  <defaultPortal>web01</defaultPortal>  <defaultPage>eduki</defaultPage>  <defaultLang>es</defaultLang>  </portalFiles>  <portalCookieName>r01hpPortalCookie</portalCookieName>  </portalServer>  <!-- Metrics see http://metrics.dropwizard.io/3.1.0/ ============================= -->  <metrics enabled=*'true'*>  <consoleReporter enabled=*'false'* reportEvery=*"30s"* />  <slf4jReporter enabled=*'true'* reportEvery=*"30s"*/>  <!-- visualVM can be used to inspect metrics: -->  <!-- 1.- Install visualVM MBeans plugin: tools > plugins > Available plugins > [VisualVM MBeans] -->  <!-- 2.- Select [Tomcat] (or whatever) and go to the [MBeans] tab -->  <!-- 3.- Using the tree go to [Metrics] -->  <!-- 4.- double-clicking at any metric value a graph can be seen -->  <jmxReporter enabled=*'false'*/>  <!-- if metrics restservices are enabled some info is available through an admin servlet (restServices) -->  <!-- METRICS: http://localhost:8080/r01hpProxyWar/r01hpMetricsRestServicesServlet/metrics -->  <!-- HEALTH-CHECK: http://localhost:8080/r01hpProxyWar/r01hpMetricsRestServicesServlet/healthcheck -->  <!-- THREADS: http://localhost:8080/r01hpProxyWar/r01hpMetricsRestServicesServlet/threads -->  <!-- PING: http://localhost:8080/r01hpProxyWar/r01hpMetricsRestServicesServlet/ping -->  <restServices>true</restServices>  </metrics>  </portalpageappembedfilter> |

The file contains the following sections:

|  |  |
| --- | --- |
| Not embedded resources | A list of regular expressions that will be matched against the requested url to see if the response should be embedded into a portal page:  <!-- Resources NOT embedded into a portal page ============================================ -->  <!-- A list of regular expressions that will be matched agains the URL path of the resource -->  <notEmbeddedResources>  <!--  <urlPathRegExp>/not-embeded/.\*</urlPathRegExp>  -->  </notEmbeddedResources> |
| Portal Server config | The portal page files location [R01HP app embed filter] will look after portal page files a the location set at:  <!-- the filesystem path where the container pages can be found -->  <root>/data/r01hp/</root>  <pages>/html/pages/portal</pages>  … so if a portal page like web01-page1 is requested, the physical page HTML file will be looked after at /data/r01hp/web01/html/pages/portal/page1  The default portal/page/lang to be used if none can be guess from the request  <!-- the default portal/page/lang to be used if none can be guess from the request -->  <defaultPortal>web01</defaultPortal>  <defaultPage>eduki</defaultPage>  <defaultLang>es</defaultLang>  The portal cookie name where the portal / page / lang will be looked after if none can be guessed from the url  <portalCookieName>r01hpPortalCookie</portalCookieName>  [R01HP app embed filter] caches the portal pages in memory; the cache can be configured as:  <cacheConfig>  <initialCapacity>10</initialCapacity>  <maxSize>100</maxSize>  <checkInterval>20s</checkInterval>  </cacheConfig> |
| Metrics | [R01HP app embed filter] has an embedded metrics system that can give a view of the system work |

The properties can be overridden at the web.xml in two levels:

|  |  |
| --- | --- |
| *Level 1: set the path of a custom properties file:* | <!-- [1]: properties file -->  <init-param>  <param-name>r01hp.appembed.configFor</param-name>  <param-value>xxx.myportalpageappembed</param-value>  </init-param>  The system will search the **classpath** looking after a file called  xxx.myportalpageappembed.properties.xml |
| *Level 2: set individual properties* | <!-- A list of regular expressions (separated with ;) that will be matched against the URL path of the resource -->  <init-param>  <param-name>r01hp.appembed.notEmbeddedResources</param-name>  <param-value>/not-embedded/.\*;</param-value>  </init-param>    <!-- true if codahale's metrics are enabled -->  <init-param>  <param-name>r01hp.appembed.metricsEnabled</param-name>  <param-value>true</param-value>  </init-param>    <!-- the filesystem path where the container pages can be found -->  <init-param>  <param-name>r01hp.appembed.appContainerPageFilesRootPath</param-name>  <param-value>d:/temp\_dev/r01hp</param-value>  </init-param>  <init-param>  <param-name>r01hp.appembed.appContainerPageFilesRelPath</param-name>  <param-value>/html/pages/portal</param-value>  </init-param>    <!-- the default portal/page/lang to be used if none can be guess from the request -->  <init-param>  <param-name>r01hp.appembed.defaultPortal</param-name>  <param-value>web01</param-value>  </init-param>  <init-param>  <param-name>r01hp.appembed.defaultPage</param-name>  <param-value>container2</param-value>  </init-param>  <init-param><param-name>r01hp.appembed.defaultLang</param-name>  <param-value>es</param-value>  </init-param>  <!-- the container page file to be used it the requested one cannot be found (beware! this file is loaded from the classpath) -->  <init-param>  <param-name>r01hp.appembed.defaultContainerPageFileIfRequestedNotFound</param-name>  <param-value>r01hp/portal/pages/r01hpDefaultAppContainerPortalPage.shtml</param-value>  </init-param> |

… so the filter config:

* First looks after the *general* properties file (*usually located at /config/r01hp/r01hp.appembed.properties.xml*)
* Then looks after the filter’s init property named [r01hp.appembed.configFor]; if it exists, it tries to load the properties file and override the properties
* After all looks after the filter’s *fine-grained* properties; it any of them exist, they override the previously configured properties
  1. Customized R01HP AppEmbed Filter

A recommended approach to [app portal page embedding filter] usage is to **override** the filter type.

* The filter type can be used “*as is*” as previously detailed:

1. Configure the filter at the web.xml file:

<filter>

<filter-name>portalPageAppEmbedServletFilter</filter-name>

<filter-class>r01hp.portal.appembed.R01HPortalPageAppEmbedServletFilter</filter-class>

</filter>

1. Configure the filter properties:
   * + Using the r01hp general properties
     + Configuring a custom filter properties file at the web.xml file
     + Configuring individual properties at the web.xml file

* … but anyone can **create a custom filter type just extending the filter type:** r01hp.portal.appembed.R01HPortalPageAppEmbedServletFilter

Extending the filter type, gives all the flexibility to configuration configure the filter, either programmatically, either using customized properties files

|  |
| --- |
| **public** **class** CustomAppEmbedServletFilter  **extends** R01HPortalPageAppEmbedServletFilter {  **public** CustomAppEmbedServletFilter () {  **super**(*\_createAppEmbedConfig*());  }  **private** **static** R01HPortalPageAppEmbedServletFilterConfig **\_createAppEmbedConfig**() {  // this config can be loaded from a custom properties file  **return** **new** R01HPortalPageAppEmbedServletFilterConfig(  **new** R01HPortalPageAppEmbedContextDefaults(R01HPortalID.*forId*("web01"),  R01HPortalPageID.*forId*("ejeduki"),  Language.*DEFAULT*,  "r01hpPortalCookie"))  // not portal-page embedded urls  .withNotPortalEmbeddedUrlPatterns("^/my\_url\_path/.\*",  "^/my\_other\_url\_path/.\*");  }  } |

… and just configure the custom filter at the web.xml file:

<filter>

<filter-name>portalPageAppEmbedServletFilter</filter-name>

<filter-class>com.my.CustomAppEmbedServletFilter </filter-class>

</filter>

* 1. Running R01HP as a stand-alone reverse proxy
     1. Config

[R01HP] configuration is fairly easy, it’s based upon proxy config:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Proxy from the [web server] to the [portal application server]  [**Apache Web Server**]  [proxy] 🡨 this config  |  |  [**Portal App Server**]  [R01HPProxy]  |  |  [**App Server**] | The config resides at **/web/conf\_apache/conf\_common** and just includes a *proxy* for the [portal application server] like:  <Location '*/r01hProxyToPortalAppServer****X***/.\*'>  WLSRequest On  **WebLogicCluster portalAppServerHost1:port, portalAppServerHost2:port**  PathTrim /**r01hpProxyWar**  </Location>  Usually, once configured the [portal application server] there’s NO need to create any additional proxy config.  **BEWARE!** Every [portal app server] proxy identifier MUST have it’s corresponding proxy configuration (*see below*)  If a new proxy to a new [portal app server] is configured, the [web server] MUST be restarted (*graceful restart*)  On the contrary, **every time an application is deployed in production** the [portal app server] (R01HP) in charge of it’s portal page integration must be configured in a [web server] (*apache*) map that relates the [application] with the corresponding [portal app server (R01HP)]:  The map is located at:   |  |  | | --- | --- | | **Development** | /opt/apache2/conf\_platea\_nas/80/common/maps/map.apps.internet.inc | | **Testing** | **Non SSL:** /opt/apache2/conf\_platea\_nas/80/common/maps/map.apps.internet.inc  **SSL:** /opt/apache2/conf\_platea\_nas/443/common/maps/map.apps.internet.inc  **SSL-CLI:** /opt/apache2/conf\_platea\_nas/444/common/maps/map.apps.internet.inc | | **Producction** | **Non SSL:** /opt/apache2/conf\_platea\_nas/80/common/maps/map.apps.internet.inc  **SSL:** /opt/apache2/conf\_platea\_nas/443/common/maps/map.apps.internet.inc  **SSL-CLI:** /opt/apache2/conf\_platea\_nas/444/common/maps/map.apps.internet.inc |   Each map entry will correlate:   * **The application code or alias** * The proxy identifier to reach the [portal app server].   A map example:  *aplictest r01hProxyToPortalAppServer****A***  *aaa r01hProxyToPortalAppServer****B***  *bbb r01hProxyToPortalAppServer****A***  *cgi-bin\_aplic r01hProxyToPortalAppServer****B***  **BEWARE!** Any change in the applications map **DOES NOT require** a [web server] (*apache*) restart but sometimes a *graceful restart* can help. |
| Proxy from the [portal application server] to the targer [application server]  [**Apache Web Server**]  [proxy]  |  |  [**Portal App Server**]  [R01HPProxy] 🡨 this config  |  |  [**App Server**] | The config resides at **/config/r01hp/proxydefs** any file whose name matches the pattern **r01hp.xxx.proxy.conf** (where xxx can be any string) will be parsed and the proxy definitions it contains will be configured.  **BEWARE!** Any change in any of the proxy config files will **require the [R01HP] to be redeployed**  The proxy config format is almost 100% equal as the format used at the [web server] (*apache*) and since the [R01HP] is just a wrapper of the [WeblogicProxyServlet] or [WeblogicClusterProxyServlet], the config params are the ones documented at [ORACLE’s site](https://docs.oracle.com/middleware/1212/webtier/PLGWL/apache.htm#PLGWL395)  A basic config of a proxy to a **single** target [app server]:  <Location 'r01hpProxyWar/[a-zA-Z]{2}[0-9]{2}[a-z].\*(?:War|WAR)'>  WLSRequest On  **WebLogicHost localhost**  **WebLogicPort 8080**  PathTrim /r01hpProxyWar  </Location>  A basic config of a proxy to a **clustered** target [app server]:  <Location 'r01hpProxyWar/[a-zA-Z]{2}[0-9]{2}[a-z].\*(?:War|WAR)'>  WLSRequest On  **WebLogicCluster localhost:7001|localhost:7002**  PathTrim /r01hpProxyWar  </Location>  NOTE: For debugging pourposes a **simple HttpProxy** NOT weblogic proxy based can be configured like:  <Location 'r01hpProxyWar/[a-zA-Z]{2}[0-9]{2}[a-z].\*(?:War|WAR)'>  **TargetAppServerHost localhost**  **TargetAppServerPort 8080**  PathTrim /r01hpProxyWar  </Location>  **Its not recommended to use this proxy in a production environment.**  The [Weblogic] proxy can be used for **any target app server (it DOES NOT have to be a weblogic server)**, so the [weblogic] proxy can proxy requests to an [IIS] server, a PHP server or whatever  See this table bellow to know more about some common [Weblogic] proxy parameters that can be used. These parameters can be used both in sigle server and cluster configurations.   |  |  | | --- | --- | | Cookie-related | WLCookieName  IMPORTANT: In order for the filter to work correctly, the proxy container WebApp MUST set the cookie name to a name DIFFERENT to any cookie name used by any of the proxied app servers (see /WEB-INF/weblogic.xml file of r01hpProxyWar) | | TimeOuts connecting to the target app server | ConnectRetrySecs // should be less than ConnectTimeoutSecs  ConnectTimeoutSecs  WLSocketTimeoutSecs  WLRetryAfterDroppedConnection  WLIOTimeoutSecs | | Keep Alive | KeepAliveEnabled  KeepAliveSecs | | Debug | Debug // ALL, INFO...  WLLogFile // where the debug file is stored  **BEWARE!** Make sure the provided path exists and adequate permissions has been set, otherwise the application may fail to start.  DebugConfigInfo // enables the \_\_WebLogicBridgeConfig parameter to // the request query string  // (?\_\_WebLogicBridgeConfig) | | File names | DefaultFileName  ErrorPage | | POST | MaxPostSize  FileCaching // write lage POSTs to disk  WLTempDir // temp dir to write POSTs requests | | Paths | PathPrepend  PathTrim  TrimExt | | IP | WLDNSRefreshInterval  WLLocalIP | | PassThrough | WLProxyPassThrough  WLExcludePathOrMimeType | | SSL | SecureProxy  WLProxySSL  WebLogicSSLVersion  WLSSLWallet  WLProxySSLPassThrough  clientCertProxy |   Params **specific for a single-server proxy**:  WebLogicHost  WeblogicPort  Params **specific for a clustered-server proxy**:   |  |  | | --- | --- | | WebLogicCluster | Cluster instances must be separed by pipe symbol ( ‘|’ ) | | DynamicServerList |  | | Idempotent |  | | MaxSkipTime |  | | WLServerInitiatedFailover |  | | RoutingHandlerClassName |  |   **DEPRECATED PARAMS**  WLCookieName // use WLCookieName  HungServerRecoverSecs // use WLIOTimeoutSecs  **UNSUPPORTED PARAMS**  WLSRequest // apache (alternative to SetHandler weblogic-handler)  MatchExpression // cannot be set at <location> config sections  QueryFromRequest // apache  Debug // IIS  WLFlushChunks // IIS  WLSendHdrSeparately // IIS  WLForwardUriUnparsed // apache |

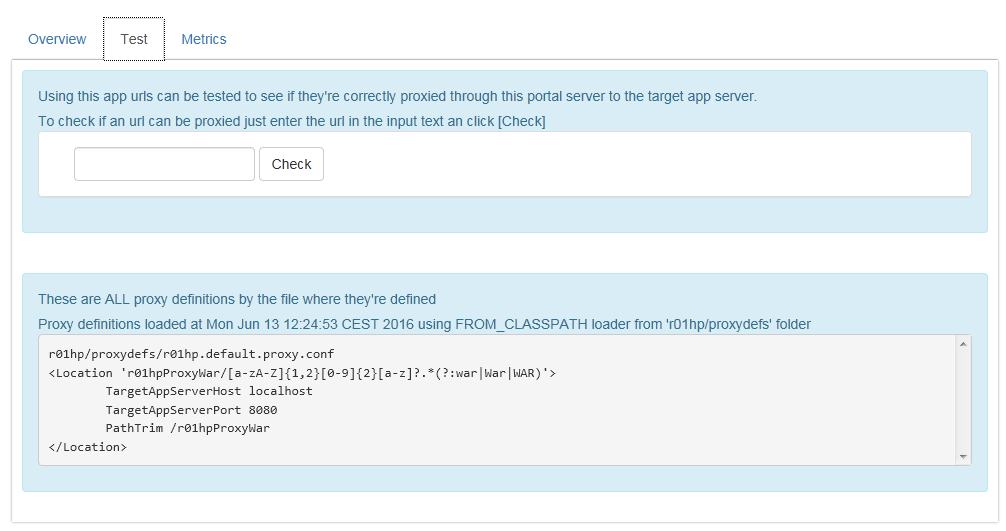
* + 1. A note for configuring applications running in IIS

IIS servers hosting applications susceptible for embedding must configure its site bindings property specifiying which virtual host will serve the application.

Additionaly HTTP compression must be disabled.

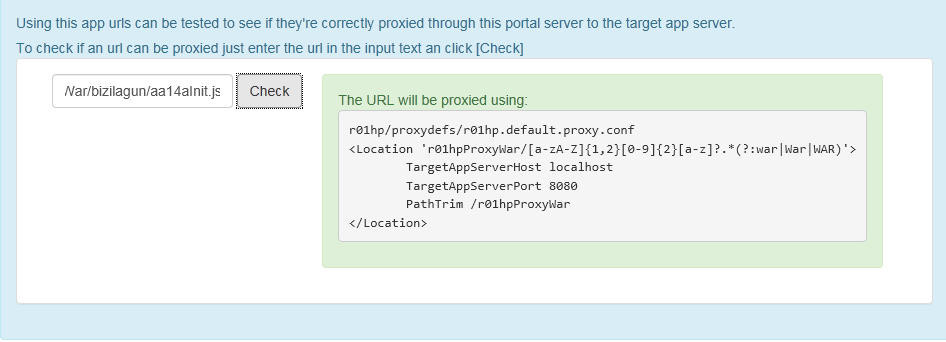
* + 1. Testing the configuration

Sometimes it’s difficult to know if a configured [application] will be proxied and what’ll be the target [application server] so ther’s a section in the console for this task:

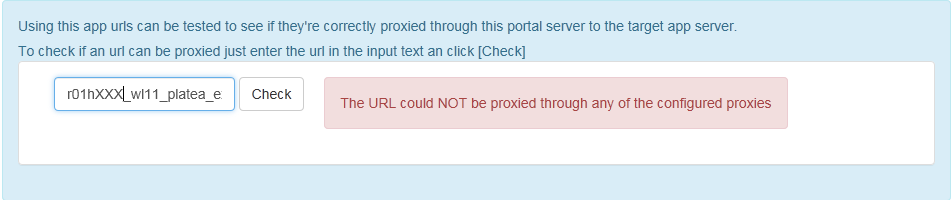


The text box provided allows to check if a certain url pattern will meet the criteria for any proxy definition configured.

The definition that will be used if found will be displayed.



If there is not a definition that will meet the criteria for the provided url a error message will be shown.

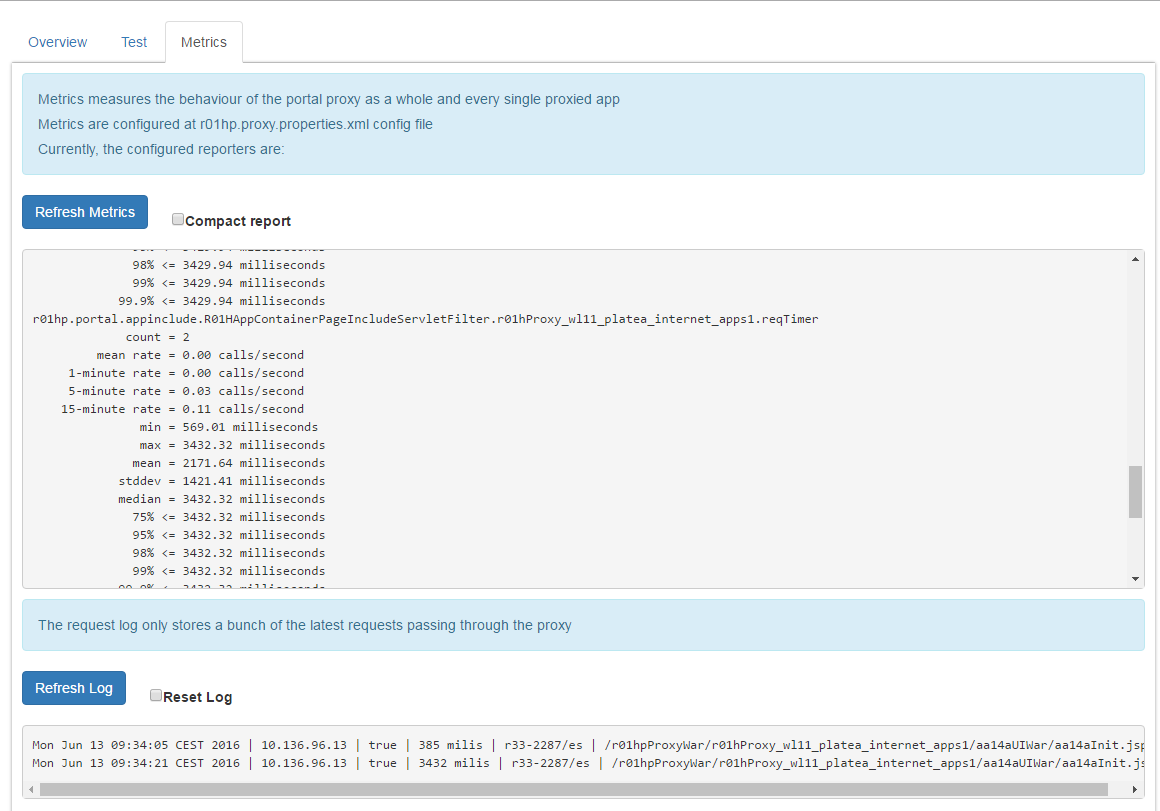


Additionaly all the current loaded proxy definitions will be listed along with the file path that contains each one.



* 1. Metrics

The system has a metrics console that offers some basic metrics about the system usage:



Avariable metrics are shown below:

* Total active requests and active request grouped by proxy configuration count
* Total HTTP 500 response served
* Count , mean, 1-minute rate,5-minute rate and 15-minute rate for the following HTTP codes: 400 (bad request), 201 (created), 204 (no content), 404 (not found), 200 (ok), 500 (server error). There is additional metrics for the remaining http codes grouped in “other” metric.
* Total and mean time elapsed for all requests and requests grouped by proxy configuration.

1. Developer useful info
   * 1. General rules

When **developing the [container portal page]:**

Here is an example of a “minimal” [container portal page]:

**Application <body> HTML will be inserted here**

**Application <head> HTML will be inserted here so they takes precedence over the page’s ones**

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="<!--#echo var='LANG'-->">

<head>

<!-- Portal static resources (for legacy [R01H] compatibility) -->

<!--#if expr="${R01\_APP\_RESOURCES} = /^true/"--><!--#include virtual='$R01\_APP\_PATH'--><!--#endif-->

**{[application HEAD]}**

**{Whatever HEAD resources (script / styles / metas)   
 includes the portal page}**

</head>

<body class='r01PrintablePage'>

<!-- ===== HEADER ====== -->

<div class="header”>

... HEADER [Visual Areas] ...

</div>

<!-- ===== BODY ====== -->

<div class="body">

...

<div id=”contentContainer" class="r01ContentContainer">

**<!--# include virtual=”$CONT” -->**

</div>

...

</div>

<!-- ===== FOOTER ====== -->

<div class="footer">

... FOOTER [Visual Areas] ...

</div>

</body>

</html>

When developing the [container page HTML], some “tricks” can be used since [container portal pages] are just apache’s SSI pages and so any [**mod\_include**](http://httpd.apache.org/docs/current/mod/mod_include.html)feature can be used, specially:

|  |  |
| --- | --- |
| Use [web server] (*apache*) environment variables | Apache [web server] sets certain environment variables that can be used in any SSI-interpreted partial (*shtml files*).  For example, in order to include an stylesheet depending on the **language**:  <!--#if expr='"$LANG" = "es"'-->  <script src=*"/appcont/xxx/jquery/jquery.ui.datepicker-es.js"*></script>  <!--#elif expr='"$LANG" = "eu"'-->  <script src=*"/appcont/xxx/jquery/jquery.ui.datepicker-****eu****.js"*></script>  <!--#endif --> |
| Include any partial file | In addition to [visual areas] (*which are a special kind of SSI partial*), any html/shtml file can be included statically in the [container portal page]:  **<!--# include virtual=”path\_to\_partial” -->**  Bear in mind that this included partial can also be interpreted by the *mod\_include* module (*if its extension is .shtml*) and so can also use [web server] (*apache*) environment variables or include another files. |

When **designing** the [application] the URL of the targeted application must meet some naming patterns (*as set by the EJIE’s hosting rules*):

|  |  |
| --- | --- |
| Java application web module | Web module URL names must start with application code, and somewhere in the path must appear the following tokens: WAR, War, Servlet or servlet.  Examples of EJIE’s hosting rules-compliant web module names are:  /{appCode}xxxWar  /{appCode}xxx/yyyServlet  Examples of EJIE’s hosting rules-NON compliant web module names are:  /{appCode}xxx  /{appCode}xxx/yyy/ |
| Non java applications (.php, .pl, .apl, .asp, .aspx, .axd, .esri) | * They must start by a letter followed by one or more digits. * Or must be at least second level in the url. |
| Some particular scenarios are taken account of for compatibility | * The URL starts with /rtta * The URL starts with /psc/ * The URL contains cgi-bin\_   (… and any other special case configured into the [web server] (apache) rules) |

When **developing** the [application]:

* Navigation links between pages of the application **can be either relative or absolute but** **NEVER should include domain names**.
* Static content should be placed in a **folder under to the common [web server] alias** ‘**appcont’** so **every portal would be able to access it**

If the static content is NOT in a folder under the ‘***appcont’*** alias, the [application] cannot be integrated / embedded in any [page] for every [portal]

As an example, referencing a resource in a certain content folder would be done as shown:

href="/appcont/[app\_content\_folder]/[resource\_relative\_path]"

* + 1. [R01HP] Filter

As said before, the new [R01HP] does NOT set any limitation to the developer so **he/she should develop the application HTML as if [R01HP] is NOT present**.

NOTE: Legacy [R01H] proxy leaved the developer limited to return a **restricted HTML** that only includes the HTML portion to be included in the <body> section wrapped in a <div> tag. **This is no longer needed, in fact, the new [R01HP] encourages the developer to return a full well-formed HTML page with all it’s <html>, <head> and <body> sections**

**BEWARE!** Once integrated / embedded in a [container portal page], the [application] visualization can be impacted by styles that the [container portal page] injects

This is specially true if the designer in charge of developing the [container portal page] did not be carefull to bear in mind that his/her styles could impact included content / applications.

Once integrated / embeded in a portal page, the [application] can take advantage of some data provided by [R01HP] proxy:

|  |  |  |
| --- | --- | --- |
| **Request parameters** | R01HPortal | Code of the portal within the application is displayed. |
| R01HPage | Code of the portal page requested. |
| R01HLang | Language code of the language versión of the portal requested. |
| **Cookies** | r01PortalInfo | **Value**: portalCode-pageCode/language  **Expiration scope**: session  **Domain**: euskadi.eus |

This data can be used by targeted aplications to know in which page from which portal are being displayed.

* + 1. Setting a non en-bedding behavior

In some cases the developer may require some HTTP request **not to be embedded**.

The [R01HP app embed filter] **will NOT embed** the target app resonse in these situations

* Any request that not allows xhtml or html content

*For example if the Accept header is JSON, XML, binary, etc*, *the response will NOT be embedded into a portal page*

* Any request whose URL is intended for a web service and contains “restService”, “restEndPoint”, “serviceEndPoint” or “webService” in the URL

NOTE that if using the [R01HP app embedded filter], additional regular expressions can be set (*se config section*)

<notEmbeddedResources>

<urlPathRegExp>/not-embeded/.\*</urlPathRegExp>

<urlPathRegExp>/also/not-embeded/.\*</urlPathRegExp>

</notEmbeddedResources>

* Any request with the parameter “R01HNoPortal=true” present. *This exception is maintained for compatibility reasons and its use is discouraged.*
  + 1. Legacy [R01H]

As said before, using the legacy [R01H] proxy, the [application] HTML must obbey certain restrictions:

1. **The application to embed** **can’t contain <html>, <head> or <body> html tags: the html code must be defined inside a <div> tag**.
2. The metas / scripts / styles or whatever content to be included in the <head> section of the portal page integration HTML result **must be defined at a file** located at:

… this is certainly a very bad restriction since every [application] page includes the same content in the <head> section