Gemini Web User Guide

2.2.6.RELEASE

Contributors:

- VMware Inc. - initial contribution
- Violeta Georgieva, SAP AG

Table of Contents

1. Overview

1.1 Introduction

Gemini Web Container implements the Web Container defined by the Web Applications Specification chapter of the OSGi Service Platform Release 4 Version 4.2 Enterprise Specification and later versions of the specification. This specification may be downloaded here.

1.2 About This Guide

This User Guide contains step-by-step instructions on how to use Gemini Web Container. This User Guide will enable you to:

- Install Gemini Web Container
- Deploy and request a simple web application
- Configure Apache Tomcat
- Configure OSGi Framework

2. Installing Gemini Web

2.1 Prerequisites

The Gemini Web Container, or GW for short, requires Java SE 6 or later to be installed. Java is available from http://www.java.com/ and elsewhere.

2.2 Installing from the ZIP Download

- Download the Equinox JAR, for example org.eclipse.osgi_3.8.1.v20120830-144521.jar, and move it to a suitable directory (e.g. ~/gemini-web-test). On the Download page, first choose the desired Release or Build, then download the JAR from the Framework section.
- Gemini Web Container is distributed as a ZIP file. Download Gemini Web Container and unzip it to ~/gemini-web-test/gemini-web.
- Configure Equinox by creating a directory ~/gemini-web-test/configuration and create a file config.ini in the configuration directory. Example file which works with 2.2.0.RELEASE is available in config.ini.zip. Essentially config.ini ensures that the dependencies of Gemini Web Container, which come in the dep directory, are installed and then the Gemini Web Container bundles are installed and started.
- Start Equinox as follows:

```
java -jar org.eclipse.osgi_3.8.1.v20120830-144521.jar -console
```

- You can then deploy WAR files (a trivial example is available in Simple-war.war.zip) and web bundles using the install and start commands from the console.

```
osgi> install file:simple-war.war
Bundle id is 40
osgi> start 40
```

- Drive the WAR or web bundle using a web browser, e.g. http://localhost:8080/simple-war should display "Hello World!".
- Stop Gemini Web Container as follows:

```
osgi> close
```

> **Tip**
> You need to stop any old instance of Gemini Web before starting it again. Otherwise, the new instance will not start correctly, because the old one still occupies the http port (and, perhaps, other system resources).

3. Configuration

3.1 Configuring the Embedded Apache Tomcat Servlet Container

Gemini Web Container embeds an OSGi-enhanced version of the Apache Tomcat Servlet Container in order to provide support for deploying Java EE WARs and OSGi *Web Application Bundles*. You configure the embedded Servlet container using the standard Apache Tomcat configuration. The main difference is that the configuration file is called `tomcat-server.xml` rather than `server.xml`. Another difference is that not all standard Apache Tomcat configuration is supported in Gemini Web Container: the restrictions are described in the remainder of this section. If you do not want to use the default settings, you can provide the `tomcat-server.xml` file located in the `$GW_HOME/config` directory.

Here's an extract of the default configuration distributed with the GW.

```
<?xml version='1.0' encoding='utf-8'?>
<Server>

  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <Service name="Catalina">

    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />

    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="localhost">

      <Host name="localhost" deployOnStartup="false" autoDeploy="false"
            unpackWARs="false" createDirs="false" appBase="">

      </Host>
    </Engine>
  </Service>
</Server>
```

Description of the Default Apache Tomcat Configuration

The following bullets describe the main elements and attributes in the default `tomcat-server.xml` file; for details about updating this file to further configure the embedded Apache Tomcat server, see the Apache Tomcat Configuration Reference.

Relative paths

If the configured path to a directory or file does not represent an absolute path, GW typically interprets it as a path relative to the `$GW_HOME` directory.

- The root element of the `tomcat-server.xml` file is `<Server>`. The attributes of this element represent the characteristics of the entire embedded Apache Tomcat servlet container.

- The `<Listener>` XML elements specify the list of lifecycle listeners that monitor and manage the embedded Apache Tomcat servlet container. Each listener class is a Java Management Extensions (JMX) MBean that listens to a specific component of the servlet container and has been programmed to do something at certain lifecycle events of the component, such as before starting up, after stopping, and so on.

- The `<Service>` XML element groups together one or more connectors and a single engine. Connectors define a transport mechanism, such as HTTP, that clients use to send and receive messages to and from the associated service. There are many transports that a client can use, which is why a `<Service>` element can have many `<Connector>` elements. The engine then defines how these requests and responses that the connector receives and sends are in turn handled by the servlet container; you can define only a single `<Engine>` element for any given `<Service>` element.

  The sample `tomcat-server.xml` file above includes two `<Connector>` elements: one for the HTTP transport, and one for the AJP transport. The file also includes a single `<Engine>` element, as required.

- The first connector listens for HTTP requests at the `8080` TCP/IP port. The connector, after accepting a connection from a client, waits for a maximum of 20000 milliseconds for a request URI; if it does not receive one from the client by then, the connector times out. If this connector receives a request from the client that requires the SSL transport, the servlet container automatically redirects the request to port `8443`.

- The second AJP Connector element represents a Connector component that communicates with a web connector via the AJP protocol.

- The engine has a logical name of `Catalina`; this is the name used in all log and error messages so you can easily identify problems. The value of the `defaultHost` attribute refers to the name of a `<Host>` child element of `<Engine>`; this host processes requests directed to host names on this servlet container.

- The `<Host>` child element represents a virtual host, which is an association of a network name for a server (such as `www.mycompany.com`) with the particular server on which Catalina is running.

- Note that multiple `<Host>` elements are not supported in Gemini Web Container.

Connector Configuration

The Gemini Web Container supports the configuration of any connector supported by Apache Tomcat. See the default configuration above for syntax examples, and for further details of the configuration properties supported for various `<Connector>` implementations, consult the official Apache Tomcat HTTP Connector documentation. For detailed instructions on how to configure Apache Tomcat's SSL support, consult the official Apache Tomcat SSL Configuration HOW-TO.

Cluster Configuration

Gemini Web Container supports standard Apache Tomcat cluster configuration. By default, clustering of the embedded servlet container is disabled, and the default configuration does not include any clustering information. See Apache Tomcat Clustering/Session Replication HOW-TO for detailed information about enabling and configuring clustering.

Default web.xml Configuration

Java Servlet specification enables web applications to provide deployment descriptor (`web.xml`) in the `WEB-INF` directory. Apache Tomcat introduces a default `web.xml` which is similar to web application's `web.xml`, but provides configurations that are applied to all web applications. When deploying a web application, Apache Tomcat uses the default `web.xml` file as a base configuration. If the web application provides its own configurations via `web.xml` (the one located in the web application's `WEB-INF`) or annotations, they overwrite the default ones. In Gemini Web Container you can also provide default configurations for all web applications. If you want to change/extend the default configurations, you can provide the default `web.xml` file located in the `$GW_HOME/config` directory.

> **Tip**
> Be careful when changing/extending the default `web.xml` as this will affect all web applications.

Here's an extract of the default configuration distributed with the GW.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <servlet>
        <servlet-name>default</servlet-name>
        <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet>
        <servlet-name>jsp</servlet-name>
        <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
        <init-param>
            <param-name>fork</param-name>
            <param-value>false</param-value>
        </init-param>
        <init-param>
            <param-name>xpoweredBy</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>3</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>/</url-pattern>
```

```
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>jsp</servlet-name>
        <url-pattern>*.jsp</url-pattern>
        <url-pattern>*.jspx</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <mime-mapping>
        <extension>abs</extension>
        <mime-type>audio/x-mpeg</mime-type>
    </mime-mapping>
    ......
    <mime-mapping>
        <extension>ppt</extension>
        <mime-type>application/vnd.ms-powerpoint</mime-type>
    </mime-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

The following bullets describe the main elements in the default `web.xml` file.

- The `<Servlet>` XML element declares a given servlet and its configurations. The sample `web.xml` file above includes two <Servlet> elements.
  - The default servlet serves static resources and processes the requests that are not mapped to any servlet. For details about default servlet configuration, see the Apache Tomcat Default Servlet Reference..
  - The jsp servlet serves the requests to JavaServer Pages. It is mapped to the URL pattern "*.jsp" and "*.jspx". For details about jsp servlet configuration, see the Apache Tomcat Jasper 2 JSP Engine..
- The `<servlet-mapping>` XML element specifies the mapping between the servlet and URL pattern.
- The `<session-config>` XML element defines the session configuration for one web application. The sample `web.xml` file above specifies that the session timeout for all web applications will be 30 minutes by default.
- The `<mime-mapping>` XML element defines a mapping between a filename extension and a mime type. When serving static resources, a "Content-Type" header will be generated based on these mappings.
- The `<welcome-file-list>` XML element specifies a list of welcome files. When a request URI refers to a directory, the default servlet looks for a "welcome file" within that directory. If the "welcome file" exists it will be served, otherwise 404 status or directory listing will be returned, depending on the default servlet configuration.

Context Configuration

Gemini Web Container supports standard Apache Tomcat web application context configuration. The Apache Tomcat Configuration Reference has a section on The Context Container which describes the mechanism that is used in GW for searching context configuration files and details the context configuration properties.

Context configuration files may be placed in the following locations, where `[enginename]` is the name of Apache Tomcat's engine ('Catalina' by default) and `[hostname]` names a virtual host ('localhost' by default), both of which are configured in `tomcat-server.xml`:

- `$GW_HOME/config/context.xml` provides the default context configuration file for all web applications.
- The `$GW_HOME/config/[enginename]/[hostname]` directory may contain:
  - The default context configuration for all web applications of a given virtual host in the file `context.xml.default`.
  - Individual web applications' context configuration files as described in the Apache Tomcat Configuration Reference. For example, the context for a web application with context path `foo` may be configured in `foo.xml`.

Note that the following context configuration features are not supported in Gemini Web Container:

- Custom class loaders.
- Specifying the context path. This is specified using the `Web-ContextPath` header in the web application's `MANIFEST.MF` file.
- Specifying the document base directory.

JNDI Resources

By default Gemini Web Container supports standard Apache Tomcat JNDI Resources handling. The Apache Tomcat JDNI Resources How-To describes in details how the JNDI resources can be configured and used.

In addition to that feature Gemini Web Container provides a possibility to switch off the standard Apache Tomcat JNDI Resources handling or to use the OSGi one. One can specify the preferred option using `-DuseNaming` with one of the following options:

- `tomcat` - the default value: this is the standard Apache Tomcat JNDI Resources handling.
- `disabled` - there is no JNDI Resources handling provided by Gemini Web Container.
- `osgi` - the OSGi JNDI Resource handling will be enabled. (Gemini Naming can be used as implementation for OSGi JNDI Services Specification) The OSGi JNDI Services Specification describes in details how JNDI can be utilized from within an OSGi framework.

One can use either `osgi URL scheme` in order to look up an OSGi service, or `java URL scheme` - a feature provided by Gemini Web Container. If `java URL scheme` is chosen then additional configuration is necessary to be provided via context.xml.

```
<Context>
<Resource name="LogService"
          type="org.osgi.service.log.LogService"
          mappedName="service/org.osgi.service.log.LogService"
          factory="org.eclipse.gemini.web.tomcat.naming.factory.OsgiServiceFactory"/>
</Context>
```

The list below describes in details the different properties:

- `name` - The name of the resource that will be created. The name should be relative to the java:comp/env context.
- `type` - The fully qualified Java class name of the resource (OSGi service) that web application will expect when it performs a lookup or when it uses @Resource annotation.
- `mappedName` - the service/s that should be looked up and the filter details if any. The syntax can be seen in OSGi JNDI Services Specification: 126.6 OSGi URL Scheme.
- `factory` - The class name for the JNDI object factory. Gemini Web Container provides a special JNDI object factory in order to be able to obtain an OSGi service.

JSP Compilation

By default Apache Tomcat compiles JSP files in web applications agains Java 1.6. In order to enable JSP compilation against Java 1.7 for your web application, additional init parameters (`compilerSourceVM` and `compilerTargetVM`) should be added for the `org.apache.jasper.servlet.JspServlet` configuration. For details about `org.apache.jasper.servlet.JspServlet` configuration, see the Apache Tomcat Jasper 2 JSP Engine. `org.apache.jasper.servlet.JspServlet` configuration can be provided with the web application's web.xml.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
        <param-name>compilerSourceVM</param-name>
        <param-value>1.7</param-value>
    </init-param>
    <init-param>
        <param-name>compilerTargetVM</param-name>
        <param-value>1.7</param-value>
    </init-param>
    <init-param>
        <param-name>fork</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>xpoweredBy</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```

Jar Scanner Configuration

The standard Jar Scanner provided by Apache Tomcat is used to scan the web application for jar files containing configuration files - TLDs or web-fragment.xml files. In addition to this functionality, Gemini Web Container provides Bundle Dependencies Jar Scanner. It is used to scan the web application bundle dependencies for such configuration files. The bundles that are treated as web application bundle dependencies are:

- The required bundles by the web application bundle.
- The bundles that provide the imported packages declared by the web application bundle.

By default the Bundle Dependencies Jar Scanner will exclude the bundles listed below from the scanning process as they do not provide TLDs and web-fragment.xml files.

- org.eclipse.osgi
- javax.servlet
- javax.servlet.jsp
- javax.el

The default behavior can be changed with Gemini Web Container property `org.eclipse.gemini.web.tomcat.scanner.skip.bundles`. The syntax is `org.eclipse.gemini.web.tomcat.scanner.skip.bundles=<bundle-symbolic-name>`

3.2 Configuring the OSGi Framework

This section provides information about configuring the OSGi framework by updating the following files in the `$GW_HOME/configuration` directory:

Table 3.1. OSGi Framework Configuration Files

Property File	Description
`config.ini`	Configures the OSGi framework properties.
`java6-server.profile`	Configures the OSGi framework profile.

Configuring OSGi Framework Properties

You specify the framework properties in the `$GW_HOME/configuration/config.ini` file. The properties relevant to users are described in the following table.

Table 3.2. Framework Properties

Property	Description
`osgi.bundles`	The comma-separated list of bundles which are automatically installed and optionally started once the system is up and running.
`osgi.java.profile`	Specifies the profile to use using a `file:` URI with default value `file:configuration/java6-server.profile`.
`osgi.bundlefile.limit`	Specifies a limit on the number of jar files the framework will keep open. The minimum value allowed is 10. Any value less than 10 will disable the bundle file limit, making the the number of jar files the framework keeps open unlimited. By default the value is 100. Increase the default value if you have many jar files in the bundle class path, expect a lot of file system operations etc.

Configuring OSGi Framework Profile

You specify the framework profile in the `$GW_HOME/configuration/java6-server.profile` file. The properties relevant to users are described in the following table.

WARNING: We advise you not to change the framework profile unless you are sure you know exactly what you are doing; updating the profile could cause GW to fail.

Table 3.3. Framework Profile Properties

Property	Description
`org.osgi.framework.bootdelegation`	This property specifies the packages which are loaded by delegation to the application class loader. Bundles can load classes belonging to these packages without importing the packages. The `.*` wildcard matches any package suffix. `java.*` is always boot delegated and must not be specified in this property. Note that the ordering of the packages does not matter.
`org.osgi.framework.system.packages`	This property specifies the packages which are

Property	Description
	exported by the system bundle. It is very occasionally necessary to extend the set, for example when configuring email logging appenders since the implementation of `javax.mail` is intimately related to the implementation of `javax.activation`.