RFC 66

RFC 66 - OSGi and Web Applications

Final

23 Pages

Abstract

This specification describes how Enterprise Web Applications written to Java Servlets and JSP specifications will be supported in OSGi. Such applications typically require services of a Web Container. The OSGi based Web Container specifies how a web application packaged as a WAR can be installed into an OSGi based runtime, and how it can integrate with OSGi services. It also specifies how an OSGi bundle can provide web content managed using the Java Sevlet 2.5 specification, in addition to being managed by OSGi from a bundle lifecycle perspective. It also describes how a OSGi based Web Container facilitates interoperability between Servlet and JSP application components with components written to other component models such as OSGi Declarative Services and Blueprint services (RFC 124).

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

`Source code is shown in this typeface.`

0.3 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial draft of RFC	Jun 13 2003	David Klein, IBM, kleind@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com Thomas Watson, IBM, tjwatson@us.ibm.com
2nd draft	Sep 17, 2003	David Klein, IBM, kleind@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com Added section 5.7 to address web descriptor support. Added Notes regarding possible changes to this RFC for possible new framework capabilities.

All Page Within This Box

3rd draft	10 Jun. 04	Thomas Watson, IBM, tjwatson@us.ibm.com Removed common JSP runtime. Updated to take advantage of RFC 70 and 71.
4th draft	11 Oct. 04	Roy Paterson, IBM, rpatersn@us.ibm.com Finished removing common JSP runtime. Grammatical fixes.
Draft 5	29 November 2004	Updated to use Declarative Service to register the WebApplication service rather than a BundleActivator. BJ Hargrave, IBM, hargrave@us.ibm.com
Draft 6	9 November 2006	Minor updates to reflect R4 has shipped and to use Map instead of Dictionary. BJ Hargrave, IBM, hargrave@us.ibm.com
Draft 7	11 November 2008	Rewrote draft after Java EE subcommittee discussions Subbarao Meduri, IBM, mkv@us.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com Graham Charters, IBM, charters@uk.ibm.com Thomas Watson, IBM, tjwatson@us.ibm.com
Draft 8	12th January 2009	Subbarao Meduri, IBM, mkv@us.ibm.com Graham Charters, IBM, charters@uk.ibm.com BJ Hargrave, IBM, hargrave@us.ibm.com Adrian Colyer, SpringSource, Adrian.colyer@springsource.com Rob Harrop, SpringSource, rob.harrop@springsource.com
Draft 9	24th February 2009	URL handler for installing WAR as bundles. Updated draft with feedback from Jan EEG f2f.
Draft 10	5th March 2009	March 2nd Review feedback. Clarification on extender interactions via use of @Resource for service reference Bugs: 1172, 1181
Draft 11	16th April 2009	Updates to interface APIs Bugs: 1170, 1216, 1181, 1196, 1273, 1130, 1195, 1172, 969, 968, 967, 1199

All Page Within This Box

Draft 12	14th May 2009	Updated to remove interface APIs, and to specify the webbundle URL scheme. Bugs 1173, 1180, 1182, 1185, 1216, 1244, 1274, 1275, 1284, 1286, 1287, 1293, 1303, 1311, 1312, 1313
Draft 13	25th June 2009	Clarify bundle life cycle sections, signed WAR support Bugs 1182, 1293, 1329, 1331, 1332, 1333, 1335, 1337, 1341, 1348, 1352, 1353, 1356
Draft 14	17th July 2009	Scope of the RFC is clarified to indicate that this specification will not supersede Http Service specification, but coexists with it. Defined WAB, added clarifications for fragment bundles. WAR transformation section is separated from WAB life cycle sections. Incorporated e-mail feedback, corrected typos. Bugs: 965, 1184, 1185, 1303, 1311, 1349, 1376 Alasdair Nottingham, Subbarao Meduri, Graham Charters
Final	24th July 2009	Updated draft with comments from July EEG F2F meeting in Dublin Subbarao Meduri

1 Introduction

The OSGi Web Container provides support for a Web Application written to Servlet 2.5 or later and JSP 2.1 or later. The currently defined OSGi Http Service in the Compendium specification does not meet the needs of Servlet specifications beyond 2.1, and does not fully support the concept of Web applications.

The OSGi Web Container specifies how a web application packaged as a WAR can be installed into an OSGi based runtime, and how it can integrate with OSGi services. It also specifies how an OSGi bundle can provide web content managed using the Java Sevlet 2.5 specification, in addition to being managed by OSGi from a bundle lifecycle perspective.

Additionally, as OSGi evolves in enterprise space and new application models are emerging that leverage OSGi capabilities, it would be necessary for Java EE application components (in particular, Web components in the context of this RFC) to interoperate with other component models. Java EE components will need to access and provide OSGi services, as well as interoperate with other components written to Declarative Services and Blueprint Services (RFC 124).

While the specification provides support for OSGi programming model for web components by describing how they can participate in a service base ecosystem, it is not a goal of this specification to provide support for creating web components under a different component model such as Blueprint service. This specification is not meant to supersede existing HttpService specification, and to that end, does not define any new APIs for programmatically creating web components.

2 Application Domain

Web applications are a critical part of applications built using Java Enterprise Edition (Java EE). The Servlet/JSP model is a popular Web application development model that provides a well known API for developing applications that generate a markup language (e.g., HTML) for rendering by a suitable agent (e.g., web browser). The Java Servlet specification describes how standard web applications function in enterprise environment.

2.1.1 Structure of a Web application

A web application is a collection of resources that is typically made up of a collection of some of the following:

- Servlets
- JSPs
- Utility classes
- Static documents (HTML, images, sounds etc.)
- Resource files used by Java classes
- Descriptive meta information that ties the above elements together in `web.xml`

The following table describes the structure of a Web Application:

Web Application directory (WAR) structure

Directory	Contents
/	Visible web resources such as html, jsp, and images.
/META-INF	Contains the MANIFEST.MF manifest file, as required for jar files. An application WAR can list its dependencies on external libraries needed in the manifest as described in JAR specification. During deployment of the Web application, the Web container must make the correct versions of the extensions available to the application following the rules defined by the *Optional Package Versioning* mechanism (http://java.sun.com/j2se/1.4/docs/guide/extensions/)
/WEB-INF	Web application resources that are not visible to the outside world (i.e. can't be directly accessed by a browser's http request). This directory contains `web.xml`, the web application deployment descriptor. Note that It is not necessary for every web application to include a `web.xml` file, especially if the application only contains JSP resources and/or static resources.
/WEB-INF/classes	Java class files of the web app. Classes in this directory must be available to application classloader.
/WEB-INF/lib/	Library Jar files, which are automatically available to the web app's

	classloader. Classes in /WEB-INF/classes take precedence over classes found in WEB-INF/lib jars. Web containers must also be able to recognize declared dependencies expressed in the manifest entry of any of the library JARs under the WEB-INF/lib entry in a WAR.

Web applications can be packaged and signed into a Web Archive format (WAR) file using the standard Java archive tools. Servlet 2.5 specification provides a detailed description of the format of a WAR archive.

2.1.2 Web Application deployment descriptor (`web.xml`)

The Web application deployment includes the following types of configuration and deployment information:

- Initialization parameters for servlets
- Session configuration
- Complex mapping between URIs and servlets
- Application lifecycle listeners
- Servlet filters
- MIME type mappings
- Custom “welcome” pages
- Custom “error” pages
- Customized locale and encoding mappings
- Security role mapping

2.1.3 Web application context root

Every Web application running in a server is typically rooted at a unique base URL, also known as the context root (or context path) of the application. The server (web container) uses this initial portion of the URL to determine which web application services an incoming request.

A deployer usually specifies the context root of an application WAR at the time of deployment and allows for customizing the URL space serviced by an application. While a enterprise application has a standard way to specify application context root in a EAR (Enterprise Application Archive), currently there is no provision to specify context root of a web application inside a WAR file. The mechanism of how a context root for a WAR is supported has been vendor-specific.

2.1.4 Temporary work area

The Servlet specification requires a temporary storage directory per servlet context (application). The servlet engine must ensure that the content of temporary directory of one servlet context is not visible to servlets contexts of other applications. Given this storage requirement, this RFC mandates that OSGi implementations of this specification MUST have file system support.

2.1.5 JavaServer Pages

JSP technology provides a convenient script-like and HTML friendly mechanism that greatly simplifies dynamic content creation. JSP technology is built on top of Servlet technology, and often servlets and JSPs are deployed together in a WAR file. JSPs are compiled into Servlet classes at runtime. The Servlet and JSP specifications describe how JSPs are translated, and managed in a JSP container, and how requests are mapped to JSPs translated into servlets.

3 Problem Description

The OSGi compendium currently includes an Http Service specification. However the design of that specification was developed based upon the Servlet 2.1 specification which predated the concept of the web application in the Servlet 2.2 specification, and as such does not fully support the concept of Web applications. As specified in section 2.1.2, a typical web application can declare various application attributes declaratively in the `web.xml` file. While in theory, each of these features of `web.xml` could be implemented programmatically, enterprise developers are used to writing web applications, and there are many tools available for writing them. It would be logical that web applications should be embraced and fully supported in the OSGi environment.

In practice, the Http Service specification is not really useful for deploying more than a couple of servlets in an application. The Http Service specification also does not provide support for JSPs. We need a specification which supports the current generation of the servlet and JSP specifications and embraces the concept of web applications.

4 Requirements

This specification addresses the following requirements from RFPs 85 & 98:

1. There MUST be a standard programming approach that makes it possible to deploy a standard Java web application to a web application container programmatically. This approach may be a traditional Java programming interface registered as an object or objects in the OSGi Service Registry, it may be a programming pattern that an OSGi bundle must follow in order to ensure that it is registered as a web application via reflection, or it may be something else, such as a pattern that the bundle must use to register itself on a “whiteboard.”

2. The standard MUST NOT require that a particular configuration API or system, including the OSGi Configuration Administration Service, be supported.

3. The solution MUST support deploying web applications as WAR files, as described in the Java Servlet Specification.

4. The solution MUST allow web applications to specify which packages to import from the OSGi Framework.

5. The solution MUST allow web applications to export packages and services to the OSGi Framework, as well as allow OSGi-aware web applications to access other OSGi services.

6. The solution MUST support web applications that support Java Server Pages (JSPs).

7. The solution SHOULD support Web components to interoperate with components written to other models such as OSGi declarative services and Blueprint services. For example, the solution could support injection of component services into Java EE components based on additional meta-data associated with the application components.

8. It SHOULD be possible for a Web application bundle to remain installed when its Web Container is dynamically replaced.

9. An OSGi-compliant Web Container MUST NOT be impeded from also being compliant with the Servlet and JSP specifications.

10. An OSGi-compliant Web Container MUST support Servlets and JSPs implemented to the Servlet 2.5 and JSP 2.1 specifications or any later versions of those specifications which are backwards compatible with Servlet 2.5 and JSP 2.1.

11. The OSGi Web Container design MUST NOT require an OSGi Execution Environment greater than that which satisfies the signatures of the Servlet and JSP specifications.

12. An OSGi Web Container MAY provide additional aspects of the technology that are required for Servlet and JSP support to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by the Servlet and JSP specifications.

5 Technical Solution

5.1 Architectural overview

Bundles are the deployment and management entities under OSGi. The RFC takes a design approach where a web application is deployed as an OSGi bundle in the framework. There is exactly one bundle that corresponds to each deployed web application in the framework. This bundle will be referred to as Web Application Bundle (WAB) throughout the specification.

The specification describes the design requirements for a OSGi Web Container that supports application components written to the Servlet and JSP specifications. The Web Container itself is deployed as one or more OSGi bundles.

The design uses the OSGi extender pattern [6], where the Web container acts as a extender that is responsible for observing the life cycle of web application bundles. When a web application bundle is started, the Web container extender processes the configuration files of the application and instantiates and manages the lifecycle of Servlet and JSP components packaged in the web application. The web application bundles thus become managed bundles of the Web container extender.

5.1.1 Web Application Bundle (WAB)

A WAB is defined as a normal OSGi bundle that contains web accessible content. A WAB makes use of the Servlet and JSP programming models for providing the static and dynamic content. A web application can be

packaged as a WAB during application development. A WAB can also be created automatically at install time from a standard web application archive (WAR) as described in section 5.4.

A WAB bundle is defined as follows:

- A WAB is a valid OSGi bundle and as such must fully describe its dependencies.
- A WAB follows the OSGi bundle life-cycle.
- A WAB is differentiated from a normal bundle through the specification of the `Web-ContextPath` manifest header. The header specifies the value of the context root of the web application. All web accessible resources inside the bundle are served up relative to this path. It must begin with a forward slash '/'.

5.1.2 Fragment bundles

Fragments are bundles that can be attached to one or more host bundles by the framework, which can modify the behavior of a host bundle. A fragment can add classes to the bundle class space, the bundle entry space, and update imports and exports. Fragments are attached by the time a bundle is resolved. This specification limits what effects a fragment bundle can have on a WAB in the following way:

- A fragment can contribute additional files to be served up statically to a host bundle.
- A fragment cannot provide or replace the `web.xml` for a WAB bundle.
- A fragment can contribute classes to the bundle class space. Under Servlet 2.5, this would not allow adding servlets to a web application that are not already specified in `web.xml` deployment descriptor in the host. If future versions of Servlet specifications relax this limitation, fragments will be allowed to add servlets to a WAB.
- An OSGi Web container must only look at the headers of the host bundle to determine if a bundle is a WAB. A bundle cannot be converted to a WAB by attaching a fragment that specifies `Web_ContextPath` manifest header. Without this restriction, a bundle that is not originally authored as a WAB can potentially make parts of the bundle web accessible. This restriction will avoid such unintended consequences.

5.2 Web application life cycle

5.2.1 Installing a Web application bundle

A WAB bundle is a valid OSGi bundle, and specifies `Web-ContextPath` manifest header. A WAB bundle can be installed into the framework using the standard `BundleContext.installBundle` API variants.

A WAB can be signed, but must comply with the bundle signing rules defined in the OSGi Core Specification v4.2.

Once installed, a WAB bundle's life cycle is managed just like any other bundle in the framework.

5.2.2 Starting a web application bundle

A web application is started by starting its corresponding web application bundle. The Web container extender listens for the bundle starting, and lazy activation life cycle events to initiate this process. A bundle that has a lazy activation policy should not be transitioned to the STARTED state by the web extender unless a request is made that requires a class to be loaded. This means that a simple request to view static content should not cause a lazy activated bundle to transition to the started state, but a request to a Servlet will.

A web application and the web container extender may start in any order. The web container extender recognizes a web application bundle by looking for the presence of `Web-ContextPath` manifest header. A web container extender may not recognize a bundle as a web application unless `Web-ContextPath` header is present in its manifest.

All Page Within This Box

After recognizing a web application bundle, the extender initiates the process of deploying the application into a web container. It must generate a DEPLOYING event as described in section 5.3. The container must validate that `Web-ContextPath` manifest header value to ensure it does not match with the context path value of web applications that are currently deployed. If the context path value is not unique, the application should not be deployed, and a FAILED event must be emitted.

The Web container processes deployment information by processing `web.xml` descriptor and any deployment specific annotations specified in the application. The descriptor may specify a **`metadata-complete`** attribute, whose value, when set to `true`, indicates that the class files in the bundle need not be examined for deployment specific annotations. Annotation scanning should not prevent the ability to further weave the byte codes of the classes as required.

As web applications are modularized further into multiple bundles (and not deployed as WAR files only) it is possible that a web application bundle can have import dependencies on other deployed bundles**.** The container must fully support a `web.xml` that specifies Servlets, Filters or Listeners whose classes are obtained via an Import-Package statement. This includes any annotation scanning required by the Servlet 2.5 specification.

The Web container extender must validate the following:

- If the WAB class path specifies `WEB-INF/classes` entry, it must be the first entry in the class path.
- If the WAB contains `WEB-INF/classes` directory, then `WEB-INF/classes` must be listed as the first entry on the bundle class path.

Any validation failures must prevent the web application module from being accessible via HTTP, and must result in a FAILED event being emitted.

The container performs the necessary initialization of web components in the bundles, as described in Servlet 2.5 specification. This involves the following:

- Create `ServletContext` for the application.
- Instantiate configured `servlet event listeners`.
- Instantiate configured `application filter instances` etc.

The Web container is required to complete instantiation of listeners prior to start of execution of the first request into the application. Attribute changes to `ServletContext` and `HttpSession` objects may occur concurrently. The container is not required to synchronize the resulting notifications to attribute listener classes. Listener classes that maintain state are responsible for the integrity of the data and should handle this case explicitly.

After successfully deploying the application, a DEPLOYED event must be generated to indicate that the application is now in service.

5.2.2.1 ServletContext in Service registry

To help management agents with tracking web applications, the OSGi Web container must register the `ServletContext` of the web application bundle as a service in the service registry with `javax.servlet.ServletContext` interface and using the `BundleContext` of the web application bundle. `ServletContext` must be registered with the following service properties:

Property name	Type	Description
`osgi.web.symbolicname`	String	The symbolic name for the Web Application Bundle
`osgi.web.version`	String	The version of the Web Application Bundle. If no Bundle-Version is specified in the manifest this is not set.

All Page Within This Box

`osgi.web.contextpath`	String	The context path from which the bundles web content will be served.

5.2.3 Stopping web application bundle

A web application is stopped by simply stopping the corresponding web application bundle. In response to a bundle STOPPING event, the extender must initiate the process of undeploying the application from the web container, and clean up any resources. This will involve the following:

- An UNDEPLOYING event is emitted to signal that the application will be removed.
- Web container will stop serving content of the application.
- Web container must clean up any application specific resources. It must invoke the context listeners and destroy the servlet context of the application. The web container must notify configured listeners in the reverse order to their declaration in `web.xml`. Per servlet 2.5 specification, session listeners are notified first, followed by context listeners of the application shut down.
- Finally, an UNDEPLOYED event is emitted.

Once the bundle is stopped, the OSGi framework will automatically unregister `ServletContext` and any other services registered by the web application.

5.2.4 Uninstalling web application bundle

A web application can be uninstalled by uninstalling the corresponding web application bundle. The application bundle will be uninstalled from the OSGi framework, and will be completely removed when the framework is refreshed.

5.3 Integration with EventAdmin service

If the EventAdmin service is registered then the web container extender bundle must emit the following events:

- org/osgi/service/web/DEPLOYING – the web extender has spotted a web application bundle and started the process of deploying the web application.
- org/osgi/service/web/DEPLOYED – the web extender has finished deploying the web application, and the application is now running (in service).
- org/osgi/service/web/UNDEPLOYING – the web extender started removing the web application in response to the bundle being stopped.
- org/osgi/service/web/UNDEPLOYED – the web extender has removed the web application. The application is no longer in service.
- org/osgi/service/web/FAILED - the web extender has failed to deploy the web application, this will be fired after the DEPLOYING event has fired.

For each event the following properties must be published:

- “bundle.symbolicName” (String) the symbolic name of the web application bundle.
- “bundle.id” (Long) the id of the web application bundle.
- “bundle” (Bundle) the Bundle object of the web application bundle.
- “bundle.version” (Version) the version of the web application bundle.
- “context.path” (String) context path of the web application bundle.
- “timestamp” (Long) the time when the event occurred
- “extender.bundle” (Bundle) the Bundle object of the web container extender bundle
- “extender.bundle.id” (Long) the id of the web container extender bundle.

- "extender.bundle.symbolicName" (String) the symbolic name of the web container extender bundle.
- "extender.bundle.version" (Version) the version of the web container extender bundle.

In addition the FAILED event must also have the following property:
- "exception" (Throwable) an exception detailing the problem.

5.4 Web Application Archive (WAR) support

Servlet 2.5 specification defines WAR format for packaging web application artifacts. There already exists a large number of web applications packaged as web application archives (WARs) available today. In order to make use of these existing assets, and to help adoption of OSGi based web application, this design supports installing Web applications packaged as standard web application archive (WAR) file developed per Servlet 2.5 and JSP 2.1 specifications.

A standard WAR file needs to be transformed into a WAB before it can be installed into the framework. This requires augmenting the WAR file MANIFEST with OSGi headers such as `Bundle-SymbolicName`, `Bundle-ClassPath` etc.

This design proposes that WAR files can be installed into the OSGi framework using `BundleContext.installBundle()` API which accepts a bundle location argument, typically in the form of a URL. A special URL scheme is used for dynamically transforming a WAR file into a WAB during install. This approach has the following advantages:

1. It avoids preprocessing a WAR bundle as a necessary step before installing.
2. It allows for reusing the `installBundle` API for installing WAR files.

Bundles can be installed into OSGi framework using `BundleContext.installBundle()` API which accepts bundle location argument, typically in the form of a URL. This design proposes using a URL handler for dynamically transforming WAR file into bundle during install. This approach has the following advantages:

- It avoids preprocessing a WAR bundle as a necessary step before installing.
- It allows for reusing `installBundle` API for installing WAR files.

5.4.1 Installing WAR files

This design provides a new URL scheme which can be used to install a WAR file as a bundle. The URL scheme follows the following format:

```
webbundle:<any other url scheme>?param1=value1&param2=value2
```

The following table describes the parameters that must be supported by any `webbundle` URL handler. All these parameters are optional. The parameter names and values must be encoded as described by URL specification [8].

Parameter name	Purpose
Bundle-SymbolicName	The desired symbolic name for the bundle.
Bundle-Version	The desired version the bundle should have, this must follow the normal OSGi versioning scheme.
Bundle-ManifestVersion	The desired bundle manifest version. The only valid value for this is "2"
Import-Package	A list of packages that the war file depends on. Some packages are implicitly added. Those are not specified here.

All Page Within This Box

Web-ContextPath	The context root the web container should serve content up from.

When a connection is opened the `webbundle` URL handler opens the embedded URL and access the JAR located there, and updates the manifest as it is loaded. Any URL scheme understood by the framework can be embedded, such as an http, or file url. For instance the following is valid:

```
webbundle:https://localhost:1234/some/path
```

The `java.net.URL` object for a `webbundle` URL must return the String *webbundle* when the `getProtocol` method is called. The embedded URL must be returned in full from the `getPath` method. The parameters for processing of the manifest must be returned from the `getQuery` method.

Some forms of embedded URL also contain URL query parameters and this must be supported. Thus the value returned from `getPath` may contain a URL query. Any implementation must take care to preserve both the query parameters for the embedded URL, and for the `webbundle` URL. The following example shows an HTTP url with some query paramters:

```
webbundle:https://localhost:1234/some/path/?war=example.war?Bundle-SymoblicName=com.example
```

In this case `getPath` method of the webbundle URL must return:

```
https://localhost:1234/some/path/?war=example.war
```

5.4.2 WAR manifest processing

An implementation must register a URL handler for processing implementation specific WAR URLs. When a deployer invokes `installBundle` to install WARs using the `webbundle` URL, the URL handler gets control, and examines the input URL and adds or updates manifest headers based deployer specified options. The URL handler uses the following while computing manifest header values:

- WAR file manifest entries – developer defaults
- Properties supplied by deployer via install URL (e.g., query string) - deployer defaults
- Compute based on additional criteria, e.g., other artifacts in the archive.

The following manifest headers described below must be present in the WAB bundle manifest when the URL handler has finished the manifest enhancement of a WAR file:

1. **Bundle-ManifestVersion:**
   If the framework version is R4.2, it must be set to 2
   If `Bundle-ManifestVersion parameter` is specified in the URL, it will be used.
   If the source manifest does not have a `Bundle-ManifestVersion` header then this header MUST be added and the value should be set to 2.
2. **Bundle-SymbolicName:**
   If `Bundle-SymbolicName parameter` is specified in the URL, it will be used.
   If the source manifest does not have a `Bundle-SymbolicName` header value, the URL handler implementation must generate a valid symbolic name. How this is generated is implementation specific.
3. **Bundle-Classpath:**
   Add or update `Bundle-ClassPath` header with
   a. WEB-INF/classes/.
   b. All jars from WEB-INF/lib – order is unspecified. If these Jars declare dependencies in their manifest on other Jars in the bundle, those jars must be added to Bundle-ClassPath. The process of detecting Jar dependencies must be performed recursively.
4. **Import-Package:**

All Page Within This Box

Add or update `Import-Package` header by adding these imports if not already present.

```
javax.servlet; version=2.5
javax.servlet.http; version=2.5
javax.el; version=2.1
javax.servlet.jsp; version=2.1
javax.servlet.jsp.el;version=2.1
javax.servlet.jsp.tagext; version=2.1
```

If `Import-Package` parameter is specified in the URL, add the specified packages to Import-Package header. If a package specified by deployer options already exists in the manifest, possibly with different values for version range and attributes, it will be replaced with the one specified via deployer options.

The resulting bundle must have access to JRE packages. A system bundle exports JRE packages, and the default is to export all JRE packages. The resulting bundle may either import packages from system bundle with `resolution:=optional` or use `Require-Bundle` with system bundle. The choice is left to the implementation.

An implementation may also analyze classes in the application archive to compute the imports needed by classes and add them to Import-Package header. The Web container implementation should not require that all packages explicitly mentioned here are imported by WAB bundle, as the manifest enhancement can add precise list of packages needed by the application during analysis.

5. **Web-ContextPath:**
   Add or update `Web-ContextPath` header in the manifest.
   If `Web-ContextPath` parameter is specified in the URL, set it to the value specified.
   If the source manifest does not have a `Web-ContextPath header,` the URL handler implementation must generate a context path and set the header value to it.
   The context path must start with a leading slash (“/”), and the URL handler must add it if it is not present.

For the WAR manifest headers described below, if a corresponding URL query parameter is specified, the URL handler must generate the header and set it to the value of the query parameter. If a URL query parameter is not specified, but a header already exists in the original manifest, it must be preserved and its value must not be modified.

1. **Bundle-Version:**
   Set to `Bundle-Version` parameter specified in the URL

Manifest headers relating to signing of the WAR must be stripped out by the URL handler. See section 5.4.3 for additional discussion on signed WAR.

Any other manifest headers not described in this section must be preserved by the URL handler and must not be modified.

The URL handler must validate query parameters, and ensure that the manifest enhancements are legal. Any validation failures must result in `BundleException` and the bundle install must fail.

Once a WAB is generated and installed, its life cycle is managed just like any other bundle in the framework.

5.4.3 Signed WAR files

When a signed WAR file is installed using `webbundle` URL scheme, the manifest enhancement process could invalidate the signatures in the bundle. The OSGi specification provides support for fully signed bundles, but does not support partially signed bundles, and ignores signer data in a nested Jar file of a bundle.

All Page Within This Box

Signing WAR files is not a common practice. If verification is required, the WAR must be converted to a WAB during development. A WAB can be signed and installed as a regular bundle, and the framework will validate the signature as defined in the core specification.

5.5 Accessing OSGi environment from Web application

In order to properly integrate in a OSGi environment, a Web application may need access to OSGi service registry for publishing its services and accessing services provided by other bundles. This may require an OSGi-aware web application access to `BundleContext` object.

To facilitate this, Web container will make `BundleContext` instance available to the web application via the `ServletContext` interface using a `osgi-bundlecontext` attribute. A Servlet can obtain BundleContext as follows:

```
BundleContext ctxt = (BundleContext)ServletContext.getAttribute("osgi-
bundlecontext");
```

Alternatively, a web component can obtain `BundleContext` provided it has access to the Bundle object (e.g., `Bundle.getBundleContext()`). A general mechanism for obtaining the Bundle object from any class loaded by that bundle is being proposed through Bug 786.

5.6 JavaServer Pages support

JavaServer Pages (JSP) is a popular rendering technology that simplifies page construction. This RFC supports the JSP 2.1 specification. A web component is either a servlet or a JSP page. The servlet element in a `web.xml` deployment descriptor is used to describe both types of web components. JSP page components are defined implicitly in the deployment descriptor through the use of an implicit ".jsp" extension mapping, or explicitly through the use of a `jsp-group` element.

A unique aspect of JSPs is that the web container compiles a JSP page into a servlet class either during application deployment phase, or at the time of request processing, and dispatches the request to a servlet object of such dynamically created class. Often times, the compilation task is delegated to a separate JSP compiler component, who will be responsible for identifying the necessary tag libraries, and generating the corresponding servlet class file. The container then proceeds to load the dynamically generated class, creates a servlet instance and dispatches the request to the servlet.

Supporting in-line compilation of a JSP page inside a bundle will require that the container should maintain a private area where it can store such compiled classes. The framework provides a private persistent storage area for each installed bundle, and an implementation may leverage (but is not required to) the private storage area for this purpose. The container may keep the generated classes around persistently for the life of the application. The container may be require to construct a special classloader to load generated JSP classes such that classes from the bundle class path are visible to newly compiled JSP classes.

The JSP specification does not describe how JSP pages are dynamically compiled or reloaded. Various Web container implementations handle the aspects in proprietary ways. This specification does not bring forward any explicit requirements for supporting dynamic aspects of JSP pages.

5.7 OSGi Web Container

The specification defines an OSGi Web container implementation as one or more OSGi bundles that collectively implement Servlet 2.5 and JSP 2.1 specifications. The following section describes requirements for a OSGi web container.

5.7.1 Java SE considerations

The Servlet 2.5 specification requires J2SE 5.0 as the minimum execution environment. Consequently, it is the minimum execution environment for running the OSGi Web container.

5.7.2 Java EE considerations

An OSGi web container implementation will need to consider additional requirements in order to be Java EE compliant. Servlet 2.5, JSP 2.1 together with the Java EE 5 specification describes a comprehensive set of requirements that a Java EE compliant web container must implement.

In practice, a Web container implementation supports a subset of Java EE services and makes them available to applications, even when it is not fully compliant with Java EE specification. These include support for

- Application name spaces
- Environment naming context (e.g., `java:comp/env` access)
- Transactions

This specification highly recommends (but does not require) that a OSGi web container provide integration with

- Transaction manager (RFC 98)
- JNDI (RFC 142) - containers that need to support JNDI environment configuration from properties files must set the JNDI client's bundle's class loader to be the current Thread Context class loader prior to invoking a method on the application component. The context class loader must implement "org.osgi.framework.BundleReference" interface as described in OSGi R4.2 core specification.
- Persistence provider (RFC 143)

5.7.3 Resource lookup

When serving static resources, the web container must consider resources available in bundle space only. This should includes resources in the bundle and attached fragments. Resources in WEB-INF and META-INF folder must not be available to remote web clients. Per servlet 2.5 specification, these resources are available to the application through `ServletContext` API. Additionally, any resources available on the class path (via Import-Package, Require-Bundle, DynamicImport-Package) are only available to application classes and must not be served to remote web clients.

The `ServletContext` interface provides direct access only to the hierarchy of static content documents that are part of the Web application, including HTML, GIF, and JPEG files, via the `getResource` and `getResourceAsStream` methods of the `ServletContext` interface. These methods are not used to obtain dynamic content. For example, in a container supporting the JavaServer Pages specification, a method call of the form `getResource("/index.jsp")` would return the JSP source code and not the processed output. The full listing of the resources in the web application can be accessed using the `getResourcePaths(String path)` method. These resources will not be on the Bundle classpath, and the web container can invoke `Bundle.getEntry` and `Bundle.findEntries` methods correspondingly to service such requests on `ServletContext` interface.

5.7.4 Resource injection and annotations

The web application deployment descriptor may specify the **`metadata-complete`** attribute on the web-app element. This attribute defines whether the web descriptor is complete, or whether the classes in the bundle should be examined for annotations that specify deployment information. If **`metadata-complete`** is set to **`true`**, the container must ignore any Servlet annotations present in the class files of the application. If the full attribute is not specified or is set to **`false`**, the container should examine the class files of the application for annotations.

All Page Within This Box

An OSGi web container that is also compliant with Java EE 5 must support annotations described in SRV 14.5 of Servlet 2.5 specification. Even when a web container is not Java EE 5 compliant, this specification recommends (but not requires) supporting the following annotations:

@Resource
@Resources
@DeclareRoles
@PreDestroy
@PostConstruct
@PersistenceContext
@PersistenceContexts
@PersistenceUnit
@PersistenceUnits
@RunAs

If supported, annotations must be processed in application classes that implement the following interfaces.

- javax.servlet.Servlet
- javax.servlet.Filter
- javax.servlet.ServletContextListener
- javax.servlet.ServletContextAttributeListener
- javax.servlet.ServletRequestListener
- javax.servlet.ServletRequestAttributeListener
- javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSessionAttributeListener

5.7.5 Web application class loader

The implementation should not allow the application to override Java SE or Java EE platform classes, such as those in java.* and javax.* namespaces, that either Java SE or Java EE do not allow to be modified.

5.8 Component model interoperability

It is conceivable that a web application may need to interoperate with components written to other component models, such as a Declarative Services specification or Blueprint services (RFC 124). Per the Servlet specification, the web container owns the life cycle of a servlet. This eliminates the possibility of a servlet component to simultaneously be a component of another component model.

A typical interaction pattern is that a servlet may depend on an external service provided by a Blueprint service component. It is possible that the servlet may not be put into service until the dependent service is available, and may require injection of a service into the servlet component. This may require coordination between extenders of different component models. The OSGi component model is dynamic, and it is simply not possible to order the execution of extenders (there is no ordering in OSGi).

Interactions between different component models are facilitated by the OSGi service registry. This can managed in several ways:

- A Servlet can obtain `BundleContext` from `ServletContext`, for performing service registry operations as described in section 5.4
- Via RFC 142 and the `osgi:services` JNDI namespace
- Inject OSGi services registered as JNDI resources using `@Resource` annotation, if supported by an implementation.

The JNDI Integration specification (RFC 142) describes how OSGi services can be made available via JNDI context. It defines a "`osgi:services`" name space and leverages URL context factory pattern to facilitate JNDI integration with the OSGi service registry.

All Page Within This Box

The servlet specification requires support JSR 250 annotations for Java EE 5 compliant web containers. Of particular interest is the support for injecting resources. The `@Resource` annotation allows containers to inject JNDI resources into the components. This means, if OSGi services are accessible as JNDI resources per RFC 142 specification, services can be injected into servlet components using the resource injection mechanisms described in the servlet specification. A OSGi Web container is not required to support annotations, and this mechanism is available only in implementations that support `@Resource` annotations.

To summarize, the recommended interaction pattern between web components with other component models is as follows:

- Model component interactions as dependencies in the OSGi service registry
- Make OSGi services available via JNDI context
- Use Java EE resource injection mechanism to inject OSGi services into web components

.

5.9 Compatibility with the OSGi Http Service

An OSGi Http Service (see OSGi R4 compendium specification) may be implemented using the Web container implementation to provide compatibility with OSGi Http Service. If this implementation approach is used then an OSGi Http Service and the OSGi Web Container can co-exist on the same platform and access the same ports. Note that this implementation approach is recommended, but is not required to comply with the OSGi Web Container specification.

5.10 Web Container use of OSGi Services

The Web Container may use standard OSGi services to implement some of the features of the Servlet 2.5 specification. The following OSGi services are RECOMMENDED for implementing parts of the Servlet 2.5 specification.

- UserAdmin Service - Used to authenticate Web Application users.
- LogService - To write Servlet logs.
- SAXParserFactory - To process web descriptors.

5.10.1 Web Container use of UserAdmin

Web Application resources that have security constraints must only be accessed by authenticated users (see Servlet 2.3 specification). To authenticate users the UserAdmin service may be used. The password credentials for each user should be stored with the key "password". For example, to get the password for a user from UserAdmin the following code can be used:

```
Role role = userAdmin.getRole(username);
String password = null;
if (role instanceof User) {
            User user = (User) role;
            // get the password, Here we assume the password is stored a the
            // credential "password".
      password = (String)user.getCredentials().get("password");
}
```

When security constraints are set for a Web Application resource a role-name can be set. For example, the following web descriptor tags sets a security constraint for a whole Web Application that allows any valid OSGi user to have access:

```
<security-constraint>
            <web-resource-collection>
                        <web-resource-name>Web Log Example</web-resource-name>
                        <description>
                                    An example of a Web Application with a security-constraint
```

All Page Within This Box

```
                        </description>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <description>Any valid OSGi User</description>
                        <role-name>user.anyone</role-name>
                </auth-constraint>
</security-constraint>
```

The role-name can be an individual UserAdmin User or Role. In the above example user.anyone is the default Role that all UserAdmin Users have (see OSGi UserAdmin specification).

5.10.2 Web Container use of LogService

The OSGi LogService may be used to store any logs that servlets write. A Servlet can log messages by calling the GenericServlet.log() or the ServletContext.log() methods (see Servlet 2.3 specification). Calls to either of the log() methods may be logged to an OSGi LogService.

5.10.3 Web Container use of SAXParserFactory

The web descriptor file /WEB-INF/web.xml is an XML file (see Servlet 2.3 specification). A SAX parser may be obtained by using a SAXParserFactory service to process the XML file.

6 Requirements not considered

Following requirements are not considered for the design of this specification.

- The solution SHOULD also support deploying web applications in exploded form from a file system directory, following the format of the WAR file from the Java Servlet Specification.
- The solution MUST support deploying in-memory web applications. For this type of deployment, the bundle deploying a web application will be expected to supply an XML document conforming to the “web.xml” format defined in the Java Servlet Specification, plus a mechanism for loading classes and static resources for the web application. For instance, the interface could require a DOM tree and a *ClassLoader* object, or some other set of standard programming artifacts.

7 Security Considerations

All Page Within This Box

7.1 Web Container bundle

The Web Container should only be implemented by a trusted bundle. This bundle requires the following security permissions.

- ServicePermission[get] for the `org.osgi.service.log.LogService` interface. This allows the Web Container to log servlet messages.
- ServicePermission[get] for the `org.osgi.service.useradmin.UserAdmin` interface. The UserAdmin service can be used to implement the J2EE declarative security model.
- ServicePermission[get] for the `javax.xml.parsers.SAXParserFactory` interface. The SAXParserFactory service can be used to process Web Application Web descriptors.
- UserAdminPermission[getCredential] for the credential "password". Used to lookup user passwords for the J2EE declaritive security model.
- AdminPermission[resource,class] to call Bundle.getEntry, Bundle.getEntryPaths and Bundle.loadClass on the Web Application Bundles.

8 Document Support

8.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Servlet 2.5 specification

[3]. JSP 2.1 specification

[4]. Java EE 5 specification

[5]. RFC 124 (Blueprint services)

[6]. Kriens, P. The OSGi Extender Model, http://www.osgi.org/blog/2007/02/osgi-extender-model.html

[7]. PAX Web Extender http://wiki.ops4j.org/display/ops4j/Pax+Web+Extender

[8]. Uniform Resource Locators, RFC 1738, http://www.ietf.org/rfc/rfc1738.txt

8.2 Author's Address

All Page Within This Box

Name	Subbarao Meduri
Company	IBM Corporation
Address	4205 Miami Blvd, Durham, NC USA
Voice	+1 919 254 5210
e-mail	mkv@us.ibm.com

Name	Alasdair Nottingham
Company	IBM Corporation
Address	MP211 IBM Hursley Park, Winchester, SO21 2JN, UK
Voice	+44 1962 817416
e-mail	not@uk.ibm.com

Name	Dave Klein
Company	IBM Corporation
Address	11501 Burnet Rd, Austin, TX 78758, USA
Voice	+1 512 838 9420
e-mail	kleind@us.ibm.com

Name	BJ Hargrave
Company	IBM Corporation
Address	11501 Burnet Rd, Austin, TX 78758, USA
Voice	+1 512 838 9939
e-mail	hargrave@us.ibm.com

Name	Tom Watson
Company	IBM Corporation
Address	11501 Burnet Rd, Austin, TX 78758, USA
Voice	+1 512 838 4533
e-mail	tjwatson@us.ibm.com

Name	Roy Paterson
Company	IBM Corporation
Address	11501 Burnet Rd, Austin, TX 78758, USA
Voice	+1 512 838 8898
e-mail	rpatersn@us.ibm.com

8.3 Acronyms and Abbreviations

WAR – Web Application Archive

JSP – Java Server Page

8.4 End of Document

All Page Within This Box