RFC 189 Http Whiteboard Service

Draft

29 Pages

Abstract

The current Http Service specification is based on Servlet API 2.1. As such it misses newer functionality such as Servlet Filters or event listeners. In addition use of the service does not support the recent whiteboard pattern approach. This RFC lists requirement to create a new Http Whiteboard Service specification.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 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.`

0.6 Revision History

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

Revision	Date	Comments
Initial	11/02/12	Initial Version Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com
Update	01/27/12	Update on Feedback from Orlando F2F and BJ Hargrave on the CPEG mailing list. Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com
Update	01/28/12	Update on feedback from Austin F2F • Removal of new registration/unregistration methods • Clarification of Servlet API 3 registration methods • Definition of the osgi.whiteboard namespace • Minor clarifications and fixes Felix Meschberger, Adobe Systems Incorporated, fmeschbe@adobe.com
Update	04/16/13	Update with feedback from Cologne F2F • Annotations and asynchronous processing Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	05/22/13	Added section about listener registration Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	07/15/13	Updated with feedback from Palo Alto F2F • Updated listener handling • Clarified service lifecycle handling • Renamed “pattern” property to “path” Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	07/29/13	Updated with feedback from CPEG call • Changed handling of multiple whiteboard implementation Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com

Revision	Date	Comments
Update	08/15/13	Updated with feedback from BJ (partially already mentioned at the Palo Alto F2F) : • Clean up requirements list • Several clarifications / rewordings, samples • Moved DTOs to org.osgi.dto.service.http • Added security permissions Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	08/23/13	Update with feedback from CPEG call and add missing pieces: • use different registration properties for servlets and servlet filters • add notes about service life cycle and clarify properties for each service • Use consistent naming, changed the flow of chapters for easier reading Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	10/01/13	Update with feedback from CPEG call: • Reformat by moving common properties into separate chapter • Use prototype scope Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	10/25/13	Update with bug 2468 (RFC 180) Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	2013-11-11	API/Javadoc improvements BJ Hargrave, IBM

Revision	Date	Comments
Update	02/28/14	Update with feedback from Austin F2F – new abstract class as a replacement for HttpContext – add dispatching configuration for servlet filters – clarify mapping of ServletContext methods – allow a path configuration for contexts – added serviceId property to DTOs – Renamed ResourceServletDTO to ResourceDTO (bug 2572) – Created DTO hierarchy, context as the root (bug 2572) Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	04/03/14	– Update with feedback from CPEG call – Undeprecate HttpService and move properties from runtime to service registration properties – Remove shared attribute from ServletContextHelper – Clarify session handling – Minor clarifications Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	04/28/14	– Deprecate HttpService (again) and move service registration properties to HttpServiceRuntime. Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	05/07/14	Update with feedback from Basel F2F – Leave HttpService as is – Update DTOs to contain failed bindings – Rename specification to Http Whiteboard Service – Add chapter on potential updates to the Http Service – Move DTOs to separate package with new “root” DTO: Runtime DTO – Add new RequestInfoDTO Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com
Update	05/22/14	Add regexp support for servlet filters and change filter ordering (highest service ranking is first now) Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com

Revision	Date	Comments
Update	06/05/14	Fix minor typos David Bosschaert, Adobe Systems Incorporated, bosschae@adobe.com

1 Introduction

The OSGi Specifications currently only contain limited specification support for creating Web Applications in an OSGi context:

- Http Service Specification based on Servlet API 2.1. Apart from being based an old Servlet API version and being silent about how more recent versions are supported the main problem with this specification is that a provider of servlets and resources has to grab the Http Service first before being able to register servlets and resources. There is no whiteboard pattern support.

- Web Applications Specification basically just defines how existing web applications may be enhanced with OSGi Manifest headers and deployed into the OSGi Framework as-is. This is fine for moving existing web applications with minimal changes into the OSGi framework.

Some thoughts are already listed on the OSGi Community Wiki at http://wiki.osgi.org/wiki/WebExperience.

2 Application Domain

Developers need to use the full extend of current Servlet API specifications (as of this writing Servlet API 3.0 is the most recent version). As such there is a need to register servlet filters and event listeners.

3 Problem Description

3.1 Support for dated Servlet API 2.1

Current support for web applications using the Http Service in traditional OSGi based applications is limited to servlets and resources. From the current Servlet API 3.0 specification the following functionality is missing:

- Servlet Filters
- Servlet Event Listeners
- Asynchronous Requests

At this moment some of this missing functionality is covered in a proprietary way. Examples are the Apache Felix Http Whiteboard support or the OPS4J Pax Web collection of bundles.

3.2 Dependency on the HttpService service

Currently the HttpService service (or one of them if multiple services exist in a framework) must be accessed to be able to register servlets and/or resources. In addition to register a servlet or resource an instance of the `HttpContext` interface is required.

This makes it very cumbersome to easily register servlets and resources. Particularly it is hard to come up with an `HttpContext` instance which for example uses an authentication mechanism available in the framework to implement the `handleSecurity` method.

To reduce (or simplify) this dependency it would be helpful to just register servlets as services and have them registered with a matching Http Service in a whiteboard pattern style. Likewise registration of static resources would be supported in an extender pattern style.

At this moment some of this missing functionality is covered in a proprietary way. Examples are the Apache Felix Http Whiteboard support or the OPS4J Pax Web collection of bundles.

3.3 Configuration

The Http Service specification currently declares a number of framework properties to configure the Http Service. This raises a number of issues:

- Unable to dynamically reconfigure the Http Service in an easy way
- Incomplete configuration. For example the local interface to bind to is not an official configuration property
- When the Http Service is implemented as bridge to a Servlet Container in which the OSGi framework is deployed (e.g. as part of a Web Application) these properties have no effect.

In addition the actual configuration of an Http Service instance cannot be easily queried/introspected.

4 Requirements

4.1 New Http Whiteboard Service API

HS-1 The solution MUST provide a Http Whiteboard Service specification to refer to Servlet API 3.0 specification and define to what extent the Http Whiteboard Service provides support.

HS-2 The solution MUST provide a service API to support Servlet registration with patterns as defined by the Servlet API specification (Section 12.2, Specification of Mappings, in the Servlet API 3.0 specification). This requirement aligns servlet registration to functionality provided by the Servlet API web application descriptor (`web.xml`).

HS-3 The solution MUST provide a service API to support registration of Servlet API filters with patterns as defined by the Servlet API specification (Section 12.2, Specification of Mappings, in the Servlet API 3.0 specification) or referring to servlets by their names. This requirement aligns mapping filters to requests to functionality provided by the Servlet API web application descriptor (`web.xml`).

HS-4 The solution MUST add support for error page configuration.

HS-5 The solution MUST define how registered servlets and servlet filters are named.

HS-6 The solution MUST clarify ServletContext implementation for both standalone and bridged Http Whiteboard Service implementations.

HS-7 The solution MUST clarify the ServletContext scope of Servlet API listeners registered through the Http Whiteboard Service.

HS-8 The solution MUST define runtime attribute of the Http Whiteboard Service to reflect configuration of the service.

HS-9 The solution MUST define whiteboard registration of servlet services with the Http Whiteboard Service.

HS-10 The solution MUST define whiteboard registration of filter services with the Http Whiteboard Service.

HS-11 The solution MUST define whiteboard registration of servlet listener services with the Http Whiteboard Service.

HS-12 The solution MUST define registration of OSGi HttpContext services used for Servlet and Filter registration.

HS-13 The solution MUST define how servlets, filters, and servlet listener services are matched with Http Whiteboard Service implementations.

HS-14 The solution MUST define whiteboard registration of static resources.

HS-15 The solution MUST define whiteboard registration of error pages.

HS-16 The solution MUST define a capability for the whiteboard pattern registration in one of the standard namespaces (or a new namespace to be defined in the Chapter 135, Common Namespaces Specification). Bundles registering servlet, filter, and/or servlet listener services can then require this capability.

5 Technical Solution

The Http Whiteboard Specification consists of three parts:

- Whiteboard Registration support for servlets, servlet filters, listeners, resources and servlet context.
- Http Runtime Service to introspect the current state of the whiteboard service
- Potential Updates to the Http Service specification

The Http Whiteboard Specification provides all the functionality currently covered by the Http Service Specification Version 1.3 with the difference of using the whiteboard pattern instead of a programmatic API.

5.1 New Http Whiteboard Service API

The goal of this specification is to make the registration of more elements of the Web Application Descriptor available to OSGi applications compared to the current Http Service:

- Servlets may be registered with more than one pattern (instead of a single alias)
- Servlet filters (introduced in Servlet API 2.3)
- Error pages (introduced in Servlet API 2.2)
- Event Listener (introduced in Servlet API 2.3)

Of the remaining elements defined in the Web Application descriptors, MIME type mapping and login configuration iwill be provided through a similar concept as the `HttpContext` interface of the Http Service specification.

Resources (EJB) are not supported by the Http Whiteboard Service because these are outside of the scope of the Http Whiteboard Service and are supported by other mechanisms in the OSGi framework such as the service registry or through JNDI.

5.1.1 Servlet API Reference Version

Implementations of the Http Whiteboard Service Specification 1.0 are based on the Servlet API Specification Version 3.0. The implementation may support a higher version than Version 3.0 and must declare this through

those methods as well. The actual version supported is exposed through the `ServletContext.getMajorVersion()` and `.getMinorVersion()` methods.

5.1.2 Annotations

Annotations defined in the Servlet API Specifications must be ignored by an implementation of the Http Whiteboard Service Specification. This is to avoid class path scanning and rather going the OSGi way. In addition this avoids unwanted situations where servlets are registered just by the fact that a specific class is contained in a bundle – this could lead to the servlet registered twice, with the wrong context or registered at all.

Implementations of the Http Whiteboard Service Specification may support annotations through an additional proprietary opt-in mechanism like a manifest header or require capability.

5.1.3 Web Application Events

Starting with Servlet API 2.3 event listener interfaces have been defined to be notified of various events during the web application and request processing life cycle. The Http Whiteboard Service supports all listeners as defined in section 11.2, Event Listeners, of the Servlet API 3.0 specification [3].

5.1.4 Relationship to Servlet Container

Implementations of the Http Whiteboard Service specification will generally be backed by actual implementations of the Servlet API specification such as Apache Tomcat or Jetty. There also exist implementations which bridge into a servlet container into which the OSGi Framework has been deployed as a web application, for example the Apache Felix Http Service Bridge or the Equinox Http Service Bridge.

As such an Http Whiteboard Service implementation will live in a servlet context and all servlets, servlet filters, listeners and resources registered through the Http Whiteboard Service will be backed by the same `ServletContext`. However as explained in the next section, based on the configuration servlets, servlet filters, listeners and resources might get different `ServletContext` objects which delegate certain functionality to the backing context. In the case of a bridged usage the relationship looks like below where ServletContext A is the backing context.

```
Servlet Container 1:n
    Webapp 1:1
        ServletContext[A] 1:1
            Http Service 1:n
                ServletContextHelper 1:1
                    ServletContext[B]
```

With respect to Web Applications two areas need clarification as to how they are segregated or shared amongst the servlets, servlet filters, listeners and resources:

- `ServletContext` objects used for servlet and servlet filter initialization
- Http Sessions acquired by servlets and servlet filters through the `HttpServletRequest`

5.1.4.1 HttpContext, ServletContextHelper and ServletContext

The Http Service specification currently defines the correlation between an `HttpContext` used for Servlet (and now Filter) registration and the `ServletContext` used for the Servlet and Filter initialization as follows:

> `Servlet` objects require a `ServletContext` object. This object provides a number of functions to access the Http Service Java Servlet environment. It is created by the implementation of the Http Service for each unique `HttpContext` object with which a Servlet object is registered. Thus, Servlet objects registered with the same `HttpContext` object must also share the same `ServletContext` object.

The Servlet API 3.0 contains functionality which would require an extension of the existing `HttpContext` interface. As enhancing this interface would require a major version change on the Http Service specification and would break existing implementations of the Http Service and to make this specification independent from the HttpService specificationtthis specification introduces a new abstract class `ServletContextHelper`. Own implementations of this class must inherit from the abstract class and register themselves as `ServletContextHelper` services.

Servlets, resources, servlet filters, and listeners are registered with a `ServletContextHelper`. A `ServletContext` object is created by the implementation of the Http Whiteboard Service for each unique `ServletContextHelper` object with which a whiteboard service is registered. Thus, whiteboard services registered with the same `ServletContextHelper` object must also share the same `ServletContext` object.

The table lists all methods of the `ServletContext` interface and how these methods should be implemented:

Method	Implementation
`getClassLoader` (Servlet API >= 3.0)	This method must return the class loader of the whiteboard service. An implementation of the Http Whiteboard Service can achieve this by returning separate instances of the ServletContext to each whiteboard service. Such an instance would be a facade of the used Servlet Context but has access to the context of the bundle of the whiteboard service.
`getContextPath` (Servlet API >= 2.5)	Backed by Servlet Container and might return ServletContextHelper specific path. See 5.2.2
`getContext(String)`	Backed by Servlet Container. Always returns the backing context
`getMajorVersion()`	Backed by Servlet Container
`getMinorVersion()`	Backed by Servlet Container
`getMimeType(String)`	Backed by `ServletContextHelper`
`getEffectiveMinorVersion()`	Same as `getMinorVersion()`
`getEffectiveMajorVersion()`	Same as `getMajorVersion()`
`getResourcePaths(String)`	Backed by `ServletContextHelper`
`getResource(String)`	Backed by `ServletContextHelper`
`getResourceAsStream()`	Backed by `ServletContextHelper`
`getRequestDispatcher(String)`	See note 1.

`getNamedDispatcher(String)`	See note 1.
`getServlet(String)`	Backed by Servlet Container
`getServlets()`	Backed by Servlet Container
`getServletNames()`	Backed by Servlet Container
`log(String)`	Backed by Servlet Container
`log(Exception, String)`	Backed by Servlet Container
`log(String, Throwable)`	Backed by Servlet Container
`getRealPath(String)`	Backed by `ServletContextHelper`
`getServerInfo()`	Backed by Servlet Container
`getInitParameter(String)`	See note 2.
`getInitParameterNames()`	See note 2.
`getAttribute(String)`	Managed per `ServletContextHelper`
`getAttributeNames()`	Managed per `ServletContextHelper`
`setAttribute(String, Object)`	Managed per `ServletContextHelper`
`removeAttribute(String)`	Managed per `ServletContextHelper`
`getServletContextName()`	See note 3.
Programmatic Web Application configuration methods	See note 4.

Notes:

1. If the argument matches a servlet registered by the Http Whiteboard Service this method must be handled by the Http Service. Otherwise it must be backed by the Servlet Container.

2. In addition to the underlying ServletContext's initialization parameters, the Http Whiteboard Service exposes its own service registration properties and runtime attributes as ServletContext initialization parameters.

3. By default this method is backed by the Servlet Container. If the ServletContextHelper has a name, this name is returned.

4. These methods for programmatic registration of servlets, servlet filters, and listeners in a Servlet API 3 servlet container should throw `IllegalStateException`.

5.1.4.2 Http Sessions

HTTP Sessions are defined by chapter 7, Sessions, in the Servlet API 3.0 [3]. specification. HTTP Sessions are managed by the servlet container separately for each web application with the session ID sent back and forth between client and server as a cookie or as a request parameter. Assuming the session ID cookie, this is attached to the servlet context path.

Session handling is usually done by the servlet container outside of the Http Whiteboard Service implementation. Therefore the container manages a single session for the Http Whiteboard Service implementation. The Http

Whiteboard Service implementation must make sure to create a wrapper session object for each ServletContextHelper which manages the session attributes as a separate set for each ServletContextHelper.

5.1.4.3 Lifecycle of Request Handling Objects

When the Http Whiteboard Service receives a request it establishes the processing pipeline based on the available services (filters, servlets, and listeners) at this point of time and executes this pipeline. Between establishing the pipeline and finishing the processing, services used in this pipeline might become unregistered. It is up to the implementation of such a service whether it throws a servlet exception if it gets executed in that case or not. (This is basically the same as with the current Http Service and a servlet gets unregistered while it is processing a request).

5.1.4.4 Asynchronous Requests

If the implementation supports Servlet API 3.0 (or higher), servlets might use the asynchronous request handling feature. However as the servlet might not be available when the processing continues a servlet exception will be thrown.

A servlet or filter supporting the asynchronous mode must declare this with the appropriate service property `osgi.http.whiteboard.servlet.asyncSupported` or `osgi.http.whiteboard.filter.asyncSupported`.

5.1.5 Http Runtime Service

The Http Runtime Service provides introspection the current state of the Http Whiteboard Service with respect to used whiteboard services and failed usages of the whiteboard services.

5.1.5.1 Runtime Attributes

The Http Runtime Service implementation must define a set of runtime attributes which can be used by whiteboard services to associate themselves with a specific implementation. This is done via the `osgi.http.whiteboard.target` service property. The runtime attributes can be examined as service properties of the HttpServiceRuntime service registration. The runtime attributes should include the following attribute.

`osgi.http.endpoint`	A String+ value of Http Whiteboard Service endpoints provided as URLs e.g. http://192.168.1.10:8080/ or relative paths, e.g. **/myapp/**. Relative paths may be used if the scheme and authority parts of the URLs are not known such as in a bridged Http Whiteboard Service implementation. If the Http Whiteboard Service is serving the root context and neither scheme nor authority is known, the value of the property is "/". Each entry must end with a slash.

5.1.5.2 Configuration

The level of configurability of the Http Whiteboard Service may vary between implementations. Some implementations may allow to configure down to the interface and port level (for example the Jetty based Apache

Felix implementation) while others don't allow anything to be configured (for example a bridging implementation where configuration is done in the servlet container).

If an implementation supports configuration, such configuration should be supplied via the Configuration Admin Service.

The framework properties `org.osgi.service.http.port` and `org.osgi.service.http.port.secure` apply in the absence of configuration.

This draft explicitly does not define a standard configuration PID for the Http Whiteboard Service implementation to be used as this would prevent scalability/usual implementation patters, like using factory configurations or having multiple Http Whiteboard Service implementations at runtime.

5.1.5.3 Diagnostics

See chapter 6, Data Transfer Objects, on the diagnostic API.

5.1.6 Servlet API Exports

The Http Whiteboard Service implementation bundle is not required to export the Servlet API Java Packages. If it does so, the bundle must obey semantic versioning and support the portable Java Contracts as defined in RFC 180 [4]. The following sections list the entry for providing the contract for Servlet API 3.0 and Servlet API 2.5.

If the Servlet API is provided by another bundle, the Http Whiteboard Service implementation is a consumer of that API and should require the contract. The bundle providing the Servlet API should provide the corresponding contract.

5.1.6.1 Providing Serlvet API 3.0

```
Export-Package: javax.servlet; javax.servlet.http, javax.servlet.annotation,
    javax.servlet.descriptor; version=2.6
Provide-Capability: osgi.contract; osgi.contract=JavaServlet;
                    version:List<Version>="2.5,3.0";
  uses:="javax.servlet, javax.servlet.http, javax.servlet.annotation,
    javax.servlet.descriptor"
Providing Serlvet API 2.5
Export-Package: javax.servlet; javax.servlet.http; version=2.5
Provide-Capability: osgi.contract; osgi.contract=JavaServlet;
 version:Version=2.5; uses:="javax.servlet, javax.servlet.http"
```

5.2 Whiteboard Registration Support

With whiteboard registration support for servlets, listeners, resources, servlet filters, and ServletContextHelper services it is easy to register these web application elements without tracking the any service. The information required for the registration is provided with service registration properties.

The following table lists the common properties for whiteboard registration of servlets, listeners, resources and servlet filters. They are explained in more detailed in the next chapters.

Property	Type	Description

`osgi.http.whiteboard.context.select`	String	The value of this service property refers to a `ServletContextHelper` service. If this property is missing, the default context is used. If the property does start with a (it is used as a filter expression against the service properties of the ServletContextHelper, otherwise it is matched against the name of the ServletContextHelper. If no matching context exists , the whiteboard service is ignored. This situation should be logged with LogService for diagnosis. If more than one service matches, the one with the highest service ranking is used.
`osgi.http.whiteboard.target`	String	The value of this service property is an LDAP filter expression which selects the Http Whiteboard Service implementation to process the whiteboard service.

5.2.1 Target HttpService

Servlet, servlet filter, listener, and resource services may register with a `osgi.http.whiteboard.target` property containing a filter expression. A Http Whiteboard Service about to process a servlet, servlet filter, listener, or resource must match that filter against its runtime attributes. Only if the filter matches, the servlet, servlet filter, listener, or resource is used by the Http Whiteboard Service. For example a whiteboard service registered with the property

```
osgi.http.whiteboard.target = "(osgi.http.implementation.name=Admin)"
```

must only be used by an Http Whiteboard Service with the runtime attribute `osgi.http.implementation.name` having the value `admin`.

Without such a target property all available Http Whiteboard Services are matching. Even if a target property is used, still several Http Whiteboard Services might match. However, a servlet, listener, resource, or servlet filter service must only be used by a single Http Whiteboard Service. To prevent multiple uses a whiteboard support implementation must ensure to process such objects only with a single Http Whiteboard Service by themselves. If more than a single whiteboard support implementation is active at runtime, there is the potential that a servlet, listener, resource or servlet filter is used by more than a single Http Whiteboard Service. In this case such objects should use the target property described above making sure that not more than one Http Whiteboard Service matches the filter expression.

If more than one Http Whiteboard Service is matching and the servlet, servlet filter, resource and listener services are registered with prototype scope (see RFC 195 Service Scopes), this service will be used by all matching Http Whiteboard Services. If more than one Http Whiteboard Service is matching and the servlet, servlet filter, resource and listener services are registered with bundle scope, the service will be used by all matching Http Whiteboard Services registered by different bundles but only with one Http Whiteboard Service from the same bundle.

If more than one Http Whiteboard Service match, e.g, in the absence of the `osgi.http.whiteboard.target` property, any one Http Whiteboard Service may use the service. It is undefined which Http Service this is.

The runtime attributes of the Http Whiteboard Service using the servlet, servlet filter, listener, or resource service are exposed as ServletContext initialization parameters.

5.2.2 ServletContextHelper for servlets, servlet filters, resources, and listeners

By default the whiteboard support is associating servlets, servlet filters, listeners, and resources with the default ServletContextHelper of the targeted Http Whiteboard Service. Additional ServletContextHelper services can be made available through the whiteboard support. In this case the ServletContextHelper service must specify the `osgi.http.whiteboard.context.name` service property. This name can be referenced by a servlet, servlet filter, listener, or resource services.

If there are multiple, usable ServletContextHelper services registered with the same context name, the Http Whiteboard Service implementation must use the ServletContextHelper with the highest service ranking. This might lead to re-binding the servlet, servlet filter, listener or resource e.g. if a new usable ServletContextHelper with a higher service ranking arrives or the current used ServletContextHelper is unregistered (see section 5.2.3).

If a servlet or servlet filter is used by an Http Whiteboard Service implementation, the implementation calls the init() method of the servlet or servlet filter which gets a configuration object (`ServletConfig` or `FilterContext`) that returns a `ServletContext` object. The Http Whiteboard Service implementation is creating a `ServletContext` object for each `ServletContextHelper` it is using. Therefore servlets and servlet filters used by the same Http Whiteboard Service and referencing the same ServletContextHelper, share the ServletContext object.

Property	Type	Description
`osgi.http.whiteboard.context.name`	String+	For `ServletContextHelper` services this property is required and identifies the service when referred to by a whiteboard service. ServletContextHelper services without this property are ignored. The name must follow the symbolic name definition.
`osgi.http.whiteboard.context.path`	String	Optional property for defining an additional context path for the context.

A `ServletContextHelper` might be registered with a context path, like in the example below is the default context and two custom contexts registered with different paths.

```
Http Service 1:n
    ServletContextHelper [DEFAULT]
    ServletContextHelper [name=A, path=app-a]

    ServletContextHelper [name=B, path=app-b}
```

Assuming the root of the Http Whiteboard Service is accessible via the path /root, servlets registered with the default context helper will be registered under /root, servlets registered with helper A will be registered under /root/app-a and servlets registered with helper B will be registered under /root/app-b. Different ServletContextHelper services with different names may use the same context path property.

If a servlet context helper is registered with several names, the first one in the list is considered the official names and the other are aliases. The method `getServletContextName` will return the first name.

When a request is processed, the method `handleSecurity(final HttpServletRequest request, final HttpServletResponse response)` from the `ServletContextHelper` object is called before any request listener, filter or servlet is called. If the call to this method returns `false`, no further processing must take place.

The execution pipeline consisting of request listeners, filters and the servlet (see section 5.1.4.3) is assembled of the servlet matching the request and those listeners and filters which match the request. Listeners and filters are chained based on their service ranking, highest ranking first.

5.2.3 Lifecycle of servlets, servlet filters, resources, and listeners

If a servlet, servlet filter, resource or listener service is used by an Http Whiteboard Service implementation, the following order of actions are performed:

1. The service is get from the service registry
2. For servlets and servlet filters, init() is called

If the service is not used anymore, these actions are performed:

1. For servlets and servlet filters, destroy() is called
2. The service is released

As servlet and servlet filters services might come and go as well as `ServletContextHelper` services might come and go, the whiteboard service registration can be very dynamic. Therefore servlet and servlet filter services might transition between used by a Http Whiteboard Service implementation to not being used and back to be used. As in this case, `init()` and `destroy()` are called each time the service is used, the recommended way to register servlet and servlet filter services is to use the prototype scope. In that case a new instance is created for each usage. If the prototype scope is not used, the service should be implemented in a reentrant way and be prepared that after a call of `destroy()` a new initialization through `init()` might follow.

5.2.4 Servlet Registration

Servlets are registered with a list of patterns in the `osgi.http.whiteboard.servlet.pattern` service registration property. These patterns are defined by the Servlet API 3.0 specification [3]. in section 12.2, Specification of Mappings:

- A string beginning with a ‘/’ character and ending with a ‘/*’ suffix is used for path mapping.
- A string beginning with a ‘*.’ prefix is used as an extension mapping.
- The empty string ("") is a special URL pattern that exactly maps to the application's context root, i.e., requests of the form http://host:port/<context- root>/. In this case the path info is ’/’ and the servlet path and context path is empty string (““).
- A string containing only the ’/’ character indicates the "default" servlet of the application. In this case the servlet path is the request URI minus the context path and the path info is null.
- All other strings are used for exact matches only.

A servlet may register itself with the property `osgi.http.whiteboard.servlet.name` which can be used by servlet filters to address this servlet. If the servlet does not set this property, the servlet name defaults to the fully qualified class name of the service object. Therefore in that case it can't be directly referenced by a servlet filter. If there is more than one servlet with the same name and also associated with the same ServletContextHelper, then the servlet with the highest service ranking is used and the other servlet is ignored. The same happens if there is more than a single servlet using the exact value for a pattern within the same ServletContextHelper.

If a servlet is used by an Http Whiteboard Service implementation, the init() method of the servlet will be called. Once the servlet is no longer be used by the Http Whiteboard Service implementation the destroy() method will be called. All service registration properties starting with servlet.init. are passed as servlet init parameters to the servlet as well as all runtime attributes of the Http Runtime Service. The service registration properties have precedence over the runtime attributes.

Property	Type	Description
`osgi.http.whiteboard.servlet.name`	String	The name of a servlet. This name is used as the value of the `ServletConfig.getServletName()` method and defaults to the fully qualified name of the service object's class.
`osgi.http.whiteboard.servlet.pattern`	String+	Registration patterns for the servlet.
`osgi.http.whiteboard.servlet.asyncSupported`	Boolean	Declares whether the servlet supports asynchronous operation mode.
`osgi.http.whiteboard.servlet.errorPage`	String+	Register the servlet as an error page for error code and/or exception; the value may be fully qualified exception type or three digit HTTP status code. Any value not being a three digit number is assumed to be a fully qualified class name.
servlet.init.*	String+	Properties starting with this prefix are passed as servlet init parameters to the init method of the servlet.

5.2.5 Servlet Filter Registration

Servlet filters have been introduced into the Servlet API specification in Version 2.3 and thus far support for them has been absent in the Http Service specification. This specification adds support to register servlets filters through the whiteboard pattern. A servlet filter can be registered with path patterns like a servlet or a servlet filter may be mapped to a specific servlet by referencing the servlet's name.

A servlet filter can set the `osgi.http.whiteboard.filter.pattern` property to path patterns as defined by the Servlet API 3.0 specification [3]. in section 12.2, Specification of Mappings or a filter servlet can set the `osgi.http.whiteboard.filter.regexp` property to regular expressions matched against the path. A servlet filter can also reference servlets by name using the `osgi.http.whiteboard.filter.servlet` property. A servlet filter matches the request if at least one of the provided properties matches.

A servlet filter may register itself with the property `osgi.http.whiteboard.filter.name`. If the servlet filter does not set this property, the servlet filter name defaults to the fully qualified class name of the service object. If there is more than one servlet filter with the same name and also associated with the same ServletContextHelper, then the servlet filter with the highest service ranking is used and the other servlet filter is ignored.

The servlet filter dispatcher configuration can be set with the property `osgi.http.whiteboard.filter.dispatcher`. Allowed string values are REQUEST, ASYNC, ERROR, INCLUDE, and FORWARD. The default for a filter is REQUEST. See Java servlet specification 3.0, Chapter 6.2.5 for more information.

If a servlet filter is used by an Http Whiteboard Service implementation, the init() method of the servlet filter will be called. Once the servlet filter is no longer be used by the Http Whiteboard Service implementation, the destroy() method will be called. All service registration properties starting with filter.init. are passed as init parameters to the filter as well as all runtime attributes of the Http Runtime Service. The service registration properties have precedence over the runtime attributes.

Property	Type	Description
osgi.http.whiteboard.filter.name	String	The name of a servlet filter. This name is used as the value of the FilterConfig.getFilterName() method and defaults to the fully qualified name of the service object's class.
osgi.http.whiteboard.filter.pattern	String+	Registration property for a servlet filter to apply this filter to the url paths.
osgi.http.whiteboard.filter.servlet	String+	Registration property for a servlet filter to apply this filter to the referenced servlet.
~~o~~osgi.http.whiteboard.filter.regex~~p~~	String+	Registration property for a servlet filter to apply this filter to the url paths. The values must be regular expressions following the Java syntax defined in java.util.regex~~p~~ex.Pattern.
osgi.http.whiteboard.filter.asyncSupported	Boolean	Declares whether the servlet filter supports asynchronous operation mode.
osgi.http.whiteboard.filter.dispatcher	String+	Registration property for a servlet filter to set the associated dispatcher configuration when the filter should be called.
filter.init.*	String+	Properties starting with this prefix are passed as filter init parameters to the init method of the filter.

5.2.6 Resources

To register resources through the whiteboard an instance of the type javax.servlet.Servlet is registered as a regular servlet with the additional osgi.http.whiteboard.resource.prefix servlet registration property. The osgi.http.whiteboard.servlet.pattern property must also be specified.

Property	Type	Description
osgi.http.whiteboard.resource.prefix	String	This prefix is used to map a requested resource to the bundle's entries.

Example using DS:

```
@Component(property={"osgi.http.whiteboard.context.name=resource-context"})
public class ResourceHttpContext ~~implements~~extends ServletContextHelper{
   ...
}

@Component(service = javax.servlet.Servlet.class, scope=ServiceScope.PROTOTYPE,
   property={
     "osgi.http.whiteboard.servlet.pattern=/files/*",
```

```
    "osgi.http.whiteboard.resource.prefix=/tmp",
    "osgi.http.whiteboard.context.select=resource-context"})
public class MyResource extends HttpServlet {
   ...
}
```

5.2.7 Event Listeners

Event listeners register themselves under the interface(s) they are implementing. This specification supports:

- ServletContextListener
- ServletContextAttributeListener
- ServletRequestListener
- ServletRequestAttributeListener
- HttpSessionListener
- HttpSessionAttributeListener
-

Events are sent to all listeners registered in the OSGi service registry based on their registration properties. Each listener is associated with an ServletContextHelper as described in section 5.2.2.

The Http Whiteboard Service implementation gets the listeners from the service registry as soon as the associated ServletContextHelper is established and releases them when the ServletContextHelper is not available any more or the listener is unregistered.

5.2.7.1 ServletContextListener and ServletContextAttributeListener

The ServletContextListener receives events after the Http Whiteboard Service implementation has started and the corresponding ServletContextHelper is available and when either the ServletContextHelper becomes unavailable or the Http Whiteboard Service implementation is about to stop. A newly registered listener will be called with the `contextInitialized` method either if the ServletContextHelper is available or when the ServletContextHelper becomes available. As soon as the ServletContextHelper or the Http Whiteboard Service implementation becomes unavailable, the `contextDestroyed` method is called. The Http Service implementation holds the listener as long as the ServletContextHelper is available. ServletContextAttributeListeners are held for the same period of time.

Due to the nature of using the whiteboard pattern, a ServletContextListener or ServletContextAttributeListener might be registered after other services like servlets or filters for the same ServletContextHelper. And the listeners might also disappear before other services are unregistered. In this case the ordering of events might not be the same as in a typical web application where the listeners are always called first. If other whiteboard services require a listener to be setup before this whiteboard service is initialized, it needs to create some kind of dependency on the listener to ensure the correct ordering. For example, if Declarative Services is used to implement the whiteboard services, a servlet could have a mandatory reference to the listener service.

A Http Whiteboard Servlet implementation must process registered ServletContextListener and ServletContextAttributeListeners before any other whiteboard service.

Methods in the ServletContext object handed to the `contextInitialized` method of a registered ServletContextListener to programmatically register servlets, servlet filters, and listeners are not supported and should throw `UnsupportedOperationException`. The particular reason for not supporting these methods is the mismatch between the lifecycle of the servlet container and the lifecycle of the bundle trying to programmatically register Servlets, Filters, or Listeners.

If implementations of the Http Whiteboard Service decide to support dynamic registration through the servlet context from within the `contextInitialized` method, they should require a proprietary opt-in mechanism like a manifest header or require capability.

5.2.7.2 Supported Non-Whiteboard Listeners

The servlet specification defines some listener interfaces where the listener is not registered through the web.xml or the corresponding api. For example, the HttpSessionActivationListener is supported for objects registered as session attributes. For these types of listeners, whiteboard registration is neither required nor supported. Implementation of this specification support the following listeners:

- HttpSessionActivationListener
- HttpSessionBindingListener
- AsyncListener

5.2.8 Error Pages

A servlet can be marked to be called in case of errors, either if an exception is thrown during request processing or if a servlet uses the sendError method with a status code of 4xx or 5xx.

The service property `osgi.http.whiteboard.servlet.errorPage` can be specified on a servlet service. The property values can be an HTTP status code or the fully qualified name of an exception. If such a status code is set via sendError or such an exception is thrown, this servlet is invoked to render an error page. A servlet serving error page requests does not need to set the `osgi.http.whiteboard.servlet.pattern` service property. If it does so, the servlet can be called by using the path, but might wish to do so to serve regular requests as well.

Example:

```
@Component(service = javax.servlet.Servlet.class, scope=ServiceScope.PROTOTYPE,
   property={
     "osgi.http.whiteboard.servlet.errorPage=java.io.IOException",
     "osgi.http.whiteboard.servlet.errorPage=500"})
public class MyErrorServlet extends HttpServlet {
   ...
}
```

The above servlet is invoked if the status code 500 is sent via sendError or if an IOException occurs. In general error pages are invoked according to the rules defined in section 10.9.2 in the servlet specification.

If there is more than one error page registered for the same exception or error code within a single ServletContextHelper, the one with the highest service ranking is used.

5.3 Provided Capability

The Http Whiteboard Service implementation bundle must provide the `osgi.whiteboard` capability for “osgi.http”. For example:

```
Provide-Capability: osgi.whiteboard;
       osgi.whiteboard="osgi.http";
       uses:="javax.servlet, javax.servlet.http";
       version:Version="1.3"
```

The Http Whiteboard Service implementation must provide support for all whiteboard service types as outlined in this specification.

5.3.1 *osgi.whiteboard Namespace*

The whiteboard pattern leverages the OSGi service registry as a registry for objects. In the context of Http Whiteboard Service, servlets can be registered as services and the Http Whiteboard Service implementation uses these services to interact with the servlets.

A *Whiteboard Services Consumer* is a bundle that monitors the life cycle events of specific services to use their functionality when the specific services are active. It can use metadata (service properties) to control its functionality. *Whiteboard Services Providers*, register such services, therefore have a dependency on the Whiteboard Services Consumer that can be modeled with the osgi.whiteboard namespace. The definition for this namespace can be found in the following table and the `WhiteboardNamespace` class.

Name	Kind	M/O	Type	Syntax	Description
osgi.whiteboard	CA	M	String	symbolic-name	A symbolic name for the whiteboard services consumer. These names are defined in their respective specifications and should in general use the specification top level package name. For example, `org.acme.foo`. The OSGi Alliance reserves names that start with `osgi`.
version	CA	M	Version	version	A version. This version must correspond to the specification of the whiteboard services consumer.

Specifications for whiteboard services consumers (Http Whiteboard Service, Event Admin, etc.) should specify the values for these attributes. Whiteboard services consumers that provide such a capability should list the packages that they use in their implementation in the uses directive of that capability to ensure class space consistency. Whiteboard services consumers can consume a whiteboard services provider even if that bundle does not require the whiteboard consumer unless the specification explicitly forbids this. For example an Http Whiteboard Service could declare its capability with the following manifest header:

```
Provide-Capability: osgi.whiteboard;
   osgi.whiteboard="osgi.http";
   uses:="javax.servlet,javax.servlet.http";
   version:Version="1.3"
```

A bundle that depends on an Http Whiteboard Service implementation could require such a whiteboard consumer with the following manifest header:

```
Require-Capability: osgi.whiteboard;
  filter:="(&(osgi.whiteboard=osgi.http)(version>=1.3)(!(version>=2.0)))"
```

5.4 Potential Update of the Http Service

In order to get support for Servlet API 3.0 and support of the runtime inspection, the Http Service specification could be updated at least with the following chapters.

5.4.1 *Supported Servlet API*

The Http Service implementation can supped declaring the supported servlet API as explained in secion 5.1.6.

5.4.2 *Relationship to the Http Runtime Service*

If an Http Service implementation wants to support introspection through the Http Service Runtime service, the Http Service Runtime service must have a service registration property `osgi.http.runtime.httpservice.serviceid` containing the service id of the Http Service service.

As the DTOs contain a field with the service id of the whiteboarded services, in the case of servlets or resources registered via the Http Service, the Http Runtime Service needs to generate unique negative ids for these instances.

5.4.3 *Coexistance of the Http Service and the Http Whiteboard Service*

If an implementation implements both, the Http Service and the Http Whiteboard Service, services registered by any of those means coexist potentially within the same Http context. In the case of a clash between a service registered through the whiteboard pattern and the Http Service, the whiteboard service wins as it has a higher service ranking.

The default Http Context from the Http Service represents the same servlet context as the default ServletContextHelper from the Http Whiteboard service.

6 Data Transfer Objects

This RFC defines an API to retrieve administrative information from the Http Whiteboard Service implementation. The HttpServiceRuntime service is introduced and can be called to obtain various DTOs.

The DTOs for the various services contain the field serviceId. In the case of whiteboard services this value is the value of the service.id property of the corresponding service registration. If the Http Runtime Service supports introspection of an Http Service, for servlets and resources directly registered through the HttpService API, the Http Runtime Service implementation assigns each registration a unique negative service id starting with -1 and decreasing for each registration.

In the case of a clash, e.g. two servlets registered with the same path on the same servlet context, only the service with the highest service id is used. The service(s) with the lower service id(s) are unused. The Http Service Runtime provides DTOs for those unused services as well as failures when using a service like an exception thrown by a servlet from within the init() method in order to find setup problems.

See the JavaDoc for details.

7 Javadoc

8 Considered Alternatives

8.1 New methods to register Servlets and Filters

In addition to the proposed support for Whiteboard style registration of Servlets, Filters, Resources, HttpContexts, and error pages the Http Service API could have been extended to support programmatic support for such registration.

At the CPEG F2F in Austin it was decided that we should only offer one mechanism to register such objects. Since whiteboard pattern allows for simpler code than having to access a service to register with adding new API was dismissed.

8.2 Web Application Events

8.2.1 Limiting events

Instead of just sending web application events to all event listeners registered in the OSGi service registry it would be conceivable that listeners may register with a `osgi.http.service.target` service property which defines an LDAP filter to limit the Http Whiteboard Services sending events to the listener service.

I am not sure whether this would really be of use.

8.2.2 Event Admin Service

Servlet Events could be bridged into Event Admin Service events.

I am omitting such bridging right now because I am not sure of its use.

8.3 HTTP Sessions

The simplest implementation for HTTP Sessions would be to have a single HTTP Session backed by servlet container and thus shared amongst all Servlets and their servlet contexts. Yet, this would probably be unexpected for these applications which have separate servlet contexts and thus separate attribute value spaces but still share the same HTTP Session.

8.4 Resources

Alternatively to the proposed `Resource` servlet it might be conceivable to have the `osgi.http.whiteboard.path` and `osg.http.whiteboard.prefix` properties on an Http Context service to register resources to be served through the given Http Context. In this case the `path` property must be a prefix pattern. If we support multi-value properties, the `pattern` and `prefix` properties must provide the same number of values and they are put together by the same index; i.e. `path[0]` → `prefix[0]`, `path[1]` → `prefix[1]`, etc.

While this solution looks appealing, I am not sure, whether there is a conceptual fit between the Http Context service and the resource registration. On the other hand resources are served (resolved actually) through an Http Context, so to register resources an Http Context is always required.

8.5 Deprecated HttpService

Instead of updating the Http Service it has been decided to create a new specification for the Http Whiteboard Service and leave the Http Service as is. The new specification is split up in different packages, one for all whiteboard related stuff, one for the new Http Service Runtime and one for the new ServletContextHelper.

9 Security Considerations

Bundles that need to register a servlet, listener, resource filter, or http context must be granted ServicePermission[Interface Name, REGISTER] where interface name is the whiteboard interface the service is registered for.

Bundles that need to iterate the servlets, listeners, resources, filters, or servlet context helpers registered with the system must be granted ServicePermission[interface name, GET] to retrieve the services from the service registry.

In addition if a whiteboard service wants to be associated with a shared servlet context helper registered by another bundle, the bundle registering the whiteboard service must be granted ServicePermission[org.osgi.service.http.context.ServletContextHelper, GET].

Bundles that need to introspect the state of the Http Whiteboard Service runtime will need PackagePermission[org.osgi.service.http.runtime, IMPORT] and

ServicePermission[org.osgi.service.http.runtime.HttpServiceRuntime, GET] to obtain the HttpServiceRuntime service and access the DTO types.

10 Document Support

10.1 References

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

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. Rajiv Mordani, Java Servlet Specification Version 3.0, JSR-315, December 2009

[4]. Portable Java SE/EE Contracts, RFC 180, work in progress

10.2 Author's Address

Name	Felix Meschber
Company	Adobe Systems Incorporated
Address	Barfüsserplatz 6, 4055 Basel, Switzerland
Voice	+41 61 226 55 49
e-mail	fmeschbe@adobe.com

Name	Carsten Ziegeler
Company	Adobe Systems Incorporated
Address	Barfüsserplatz 6, 4055 Basel, Switzerland
Voice	+41 61 226 55 0
e-mail	cziegele@adobe.com

10.3 Acronyms and Abbreviations

10.4 End of Document