RFC 223: Http Whiteboard Updates

Draft

10 Pages

Abstract

Updates to Http Whiteboard for Release 7.

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 10.1.

`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	22.04.2016	Initial draft Carsten Ziegeler, Adobe
0.1	28.06.2016	Updates from the Darmstadt F2F Carsten Ziegeler, Adobe

1 Introduction

This RFC collects a numbers of requested enhancements to Http Whiteboard Service that were suggested after Release 6 design work was completed.

2 Application Domain

The Http Whiteboard Specification was first released in 2015 as part of Release 6. From the Version 1.0 spec:

> The OSGi Http Whiteboard Specification provides a light and convenient way of using servlets, servlet filters, servlet listeners and web resources in an OSGi environment through the use of the [7] Whiteboard Pattern.

3 Problem Description

3.1 Whiteboard Services and Http Service (Bug 2872)

If a Http Whiteboard implementation is also implementing the Http Service, the whiteboard specification does not specify whether the Http contexts for the Http Service are represented as ServletContextHelper services. There is no way for a whiteboard service to be registered in a Http Context of the Http Service. For example adding a servlet filter for all servlets managed by the Http Service is not possible.

3.2 Reusable Logic across Servlet Contexts (Bug 2900)

Servlet filters are run after ServletContextHelper.handleSecurity, therefore code for logging all requests or handling common security problems (e.g handling Cross-Origin Resource Sharing) needs the be run as part of the ServletContextHelper.handleSecurity method. There is currently no way to share this common logic across different ServletContextHelpers.

3.3 Multipart Configuration Handling (Bug 2870)

There's no way to set multipart configurations for servlets. As such there's no way to use the Servlet 3.0 file upload API.

The Servlet 3.0 way of doing this would have been:

A) using the @MultipartConfig annotation:
https://docs.oracle.com/javaee/6/api/javax/servlet/annotation/MultipartConfig.html

b) using the web.xml and providing a sub element multipart-config.

```
<servlet>
  <servlet-name>StudentRegistrationUsn</servlet-name>
  <multipart-config>
    <max-file-size>10485760</max-file-size>
    <max-request-size>20971520</max-request-size>
    <file-size-threshold>5242880</file-size-threshold>
```

```
    </multipart-config>
</servlet>
```

3.4 Support Servlets without a pattern (Bug 2897)

Version 1.0 of the Http Whiteboard Specification requires that a registered servlet has pattern. However the servlet spec allows to register named servlets which can be targeted by “named dispatching” and these servlets might be registered without a pattern.

4 Requirements

HW-0010 - Provide a way to register servlets, filters, listeners, and resources through the whiteboard service with the Http Service.

HW-0020 - Provide a mechanism to share logic between ServletContextHelpers.

HW-0030 - Provide a mechanism to configure servlets for file upload.

HW-0040 - Allow to register servlets with just a name (and no patterns)

HW-0050 – Provide a mechanism to provide an alternative logger for logging through ServletContext.

5 Technical Solution

5.1 Whiteboard Services and Http Service

As the Http Whiteboard Specification does not specify if and how Http Contexts managed by the Http Service are registered as ServletContextHelper services, there is currently no way for a whiteboard service to target these. When a servlet or resource is registered with the Http Service, it is either registered with the default Http Context or with a provided one. These objects have no way to identify them for example via a name or a path.

A whiteboard service which should be registered with an Http Context from the Http Service can target this by filtering for ServletContextHelper services having the service registration property `osgi.http.whiteboard.context.httpservice`. The value for this property is not further specified.

The following example registers a servlet filter for all servlets managed by the Http Service:

```
@Component(service = Filter.class, scope=ServiceScope.PROTOTYPE,
   property={
     "osgi.http.whiteboard.filter.pattern=/",
     "osgi.http.whiteboard.context.select=" +
            "(osgi.http.whiteboard.context.httpservice=*)"})
public class MyFilter implements Filter
```

It is up to the implementation on how the ServletContextHelper services for the Http Context are handled. It is not required that these are actually registered with the service registry, the matching might be done internally by the implementation. As the above filter might match more than one ServletContextHelper, it should be registered with the prototype scope as outlined in the Http Whiteboard Specification.

In the same way~~, servlets, resources,~~ error pages and listeners can be associated with the Http Contexts managed by the Http Service. As the Http Service defines that the first servlet or resource for a path wins, this isn't compatible with the way the whiteboard implementation would handle that case. Instead of defining various special cases to handle this, servlets and resources can't be associated with an Http Context managed by the Http Service. If it is, this is handled as an error.

~~However for servlets and resources additional special rules apply: the Http Service defines that the servlets and resources share a single namespace and the first registration for a pattern wins. In contrast the Http Whiteboard specification defines that the servlet or resource with the highest ranking wins. To not break the contract of the Http Service, for the Http Contexts representing the space of the Http Service, the rule of the Http Service Specification is enforced: if there is already a servlet or resource for a pattern, this one is continued to be used. The whiteboard service is ignored. This should be logged as a warning with the log service if available. Likewise if a servlet or resource is unregistered from these contexts, other whiteboard services with the same pattern that have previously being ignored are not registered automatically. Any servlet or resource appearing later in time might be registered.~~

~~Question (CZ): Is the special casing worth it? The only other option is to not support servlets or resources – which is another special case. We have the use case for registering whiteboard servlets with the Http Service.~~

5.2 Request Preprocessing

A new service markre interface Preprocessor allows to register services using a whiteboard pattern. The interface extends the Filter interfaces. Services of this type are always run before request dispatching is performed. If there are several services of this type, they are run in order of there service ranking, the one with the highest ranking is used first. In the case of a service ranking tie, the servlet filter with the lowest service.id is processed first.

The preprocessor is handled in the same way as filters, e.g init and destroy are called etc. However as these preprocessors are run before dispatching and therefore the targetted servlet context is not known yet, FilterConfig.getServletContext returns the servlet context of the backing implementation. The same context is returned by the request object. The context path is the context path of this underlying servlet context.

~~The service interface is modeled after the Servlet Filter interface:~~

```
@ConsumerType
public interface Preprocessor extends Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
                PreprocessorChain chain)
    throws IOException, ServletException;
}
```

The passed in chain can be used to invoke the next preprocessor in the chain, or if the end of that chain is reached to start dispatching of the request. A preprocessor might decide to terminate the processing and directly generate a response.

5.3 Multipart Configuration Handling

Support for multipart configuration handling is requested by setting the `osgi.http.whiteboard.servlet.multipartEnabled` equal to `true` for servlets or `osgi.http.whiteboard.filter.multipartEnabled` equal to `true` for filters.

???? An Http Service which does not support multipart configuration should reject such servlets and create a `FailedServletDTO` who's `failureReason` field is set to `DTOConstants.FAILURE_REASON_MULTIPART_NOT_SUPORTED`:

```
/**
 * The service is not registered as multipart support is not provided by the
 * http service.
 */
public static final int FAILURE_REASON_MULTIPART_NOT_SUPORTED = 8;
```

Multipart configuration can be refined with the following properties:

- `osgi.http.whiteboard.filter.multipart.location` (String)
- `osgi.http.whiteboard.filter.multipart.maxFileSize` (long)
- `osgi.http.whiteboard.filter.multipart.maxRequestSize` (long)
- `osgi.http.whiteboard.filter.multipart.fileSizeThreshold` (int)

???? It may not be possible for implementations to support the find grained configuration. So how should we spec this?

5.4 Support Servlets without a pattern

The requirements for a whiteboard servlet in section 140.4 is changed from requiring a configured pattern using the property `osgi.http.whiteboard.servlet.pattern`. The servlet must have at least one valid value for one of these properties:

- `osgi.http.whiteboard.servlet.pattern`
- `osgi.http.whiteboard.servlet.name`
- `osgi.http.whiteboard.servlet.errorPage`

5.5 ~~Capabilities~~

~~The Http Service implementation bundle must provide the osgi.implementation capability with name osgi.httpservice. This capability can be used by provisioning tools and during resolution to ensure that a Http Service is present. The capability must also declare a uses constraint for the org.osgi.service.http and servlet api packages and provide the version of this specification:~~

```
Provide-Capability: osgi.implementation;
                    osgi.implementation="osgi.httpservice";
                    uses:="org.osgi.service.http,javax.servlet,javax.servlet.http";
                    version:Version="1.3"
```

~~This capability must follow the rules defined for the osgi.implementation Namespace.~~

~~The bundle providing the Http Service must provide a capability in the osgi.service namespace representing this service. This capability must also declare a uses constraint for the org.osgi.service.http and the servlet api packages:~~

```
Provide-Capability: osgi.service;
```

~~objectClass:List<String>="org.osgi.service.http.HttpService";~~
~~uses:="org.osgi.service.http,javax.servlet,javax.servlet.http"~~

~~This capability must follow the rules defined for the osgi.service Namespace.~~

5.6 ~~Support for ServletContext logging~~

~~If a web component calls one of the various log() methods on the `ServletContext`, the logging is done through the implementation of the Http Whiteboard which might defer to the logging of the application server in bridged mode. In some cases its useful to log these log statements within the context of the web application.~~

~~A new method `boolean log(String, Throwable)` is added to the `ServletContextHelper`. The default implementation does nothing and returns `false`.~~

~~The implementation of the ServletContext#log methods - provided by the Http Whiteboard implementation – first calls the above log method of the ServletContextHelper. If it returns false, it continues with logging through the currently available mechanism. If it returns true, this means the ServletContextHelper took care of logging, and the Http Whiteboard implementation does not log that statement anymore.~~

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

8 Considered Alternatives

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

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

10.2 Author's Address

Name	Carsten Ziegeler
Company	Adobe Systems Incorporated

10.3 Acronyms and Abbreviations

10.4 End of Document