RFC 189 Http Service Updates

Draft

56 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 update the Http Service specification as well as possible create new specification for extended Web Applications in the context of OSGi.

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

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 Serlvet 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 be queried/introspected.

4 Requirements

4.1 Update to Http Service API

HS-1 The solution MUST update the Http Service specification to refer to the latest Servlet API specification and define to what extend the Http Service provides support.

HS-2 The solution MUST extend the Http Service 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 extend the Http Service 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 in the Http Service for both standalone and bridged Http Service implementations.

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

HS-8 The solution MUST define ~~service registration properties~~runtime attribute ~~for~~of the Http Service to reflect configuration of the service.

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

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

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

HS-14 The solution MUST define whiteboard registration of static resources with the Http Service.

HS-15 The solution MUST define whiteboard registration of error pages with the Http Service.

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 Service Update consists of two parts:

- Updates and clarifications to the the Http Service API and specification itself.
- Whiteboard Registration support for servlets, servlet filters, listeners, resources and HttpContexts.

5.1 Update Http Service API

The goal of the Http Service update is to make the registration of more elements of the Web Application Descriptor available to OSGi applications:

- 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 descriptiors, MIME type mapping and login configuration is already available through the `HttpContext` interface.

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

Registration of those elements is possible following the whiteboard pattern. Registration of servlets and resources through the Http Service is deprecated.

5.1.1 Servlet API Reference Version

Implementations of the Http Service Specification 1.3 is based on the Servlet API Specification Version 3.0. Implementatios of the Http Service Specification 1.3 may support a previous version of the Servlet API Specification only. 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 Service Specification. This is to avoid class path scanning and 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 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 live cycle. The Http 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 Service specification will generally be backed by actual implementations of the Servlet Servlet API specification such as 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 Service implementation will live in a servlet context and all servlets, servlet filters, listeners and resources registered through the Http Service will be backed by the same `ServletContext`.

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

Method	Implementation
`getClassLoader` (Servlet API >= 3.0)	### ? There can be a shared HttpContext so the whiteboard servlet can come from a different bundle than the HttpContext. ### CZ I'm not sure what the use case in an OSGi environment might be, so I tend to not support this method. Returning the class loader of the HttpContext does not provide any benefit. We could return the class loader for the http api as this exposes public api. Not sure if this provides any benefit
`getContextPath` (Servlet API >= 2.5)	Backed by Servlet Container
`getContext(String)`	Backed by Servlet Container
`getMajorVersion()`	Backed by Servlet Container
`getMinorVersion()`	Backed by Servlet Container
`getMimeType(String)`	Backed by `HttpContext`
`getEffectiveMinorVersion()`	Same as `getMinorVersion()`
`getEffectiveMajorVersion()`	Same as `getMajorVersion()`
`getResourcePaths(String)`	Backed by ~~`HttpContext`~~`Servlet Container`
`getResource(String)`	Backed by `HttpContext`
`getResourceAsStream()`	Backed by `HttpContext`

`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
`getRealPatch(String)`	Backed by ~~`HttpContext`~~`Servlet Container`
`getServerInfo()`	Backed by Servlet Container
`getInitParameter(String)`	See note 2.
`getInitParameterNames()`	See note 2.
`getAttribute(String)`	Managed per `HttpContext`
`getAttributeNames()`	Managed per `HttpContext`
`setAttribute(String, Object)`	Managed per `HttpContext`
`removeAttribute(String)`	Managed per `HttpContext`
`getServletContextName()`	See note 3.
Programmatic Web Application configuration methods	See note 4.

Notes:

1. If the argument matches a servlet registered by the Http 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 Service exposes its own service registration properties as ServletContext initialization parameters.

3. By default this method is backed by the Servlet Container. Http Service implementations may opt to implement this method in an implementation specific way such as returning a name for the Http Service.

4. These methods for programmatic registration of servlets, servlet filters, and listeners in a Servlet API 3 servlet container always throw `UnsupportedOperationException`. These methods can only be called in ServletContextListener.contextInitialized methods for listeners managed by the servlet container itself.

5.1.4.2 Http Sessions

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.

Implementations of the Http Service must ensure HTTP Sessions are not shared amongst Servlets registered with different servlet contexts. The implementation must make sure to create and destroy the sessions. HTTP Sessions are defined by chapter 7, Sessions, in the Servlet API 3.0 [3]. specification.

5.1.4.3 Lifecycle of Request Handling Objects

When the Http 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 appropriatea service property `osgi.http.whiteboard.servlet.asyncSupported`~~.~~ or `osgi.http.whiteboard.filter.asyncSupported.`

5.1.5 Http Service

The HttpService interface is fully deprecated since all the methods have been deprecated and replaced by whiteboard services.

5.1.5.1 ~~Service Registration Properties~~Runtime Attributes

The Http Service implementation must define a set of runtime attribute 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. ~~information the following exposemust~~ The runtime attributes can be examined via the HttpServiceRuntime.getAttributes ~~through its servlet registration properties:~~method. The runtime attributes should include the following attribute.

~~`osgi.http.service.`~~`osgi.http.endpoint`~~`s`~~	A String+ ~~property~~va~~l~~ue listing URLs of bound ports; e.g. http://192.168.1.10:8080/. The relevant information contained in the URLs is the scheme, bound interface and port and the (optional) context path in a Servlet API servlet container the Http Service is registered.

The port and address ~~properties~~information may not always be available to the Http Service implementation, particularly in a bridged implementation. In such cases the~~se properties~~ osgi.http.endpoint attribute may be ~~may be omitted from the service registration~~ absent.

CZ Does this mean these attributes are not available as service registration properties?

5.1.5.2 Configuration

The level of configurability of the Http 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 implementtion) 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.

5.1.5.3 Diagnostics

See chapter 6, Data Transfer Objects, on the diagnostic API. This API only allows for inspection of registered Servlets, resources, Filters, and error page locations.

5.1.6 API Version

The Http Service API version is incremented to 1.3.

5.1.7 Servlet API Exports

The Http 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 Service implementation is a consumer of that API and should require the contract. The bundle providing the Servlet API should provide the corresponding contract.

5.1.7.1 Providing Serlvet API 3.0

```
Export-Package: javax.servlet; javax.servlet.http; version=2.6
Provide-Capability: osgi.contract; osgi.contract=JavaServlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http, javax.servlet.annotation,
 javax.servlet.descriptor",
 osgi.contract;
 osgi.contract=JavaServlet; version:Version=2.5;
 uses:="javax.servlet,
 javax.servlet.http"
```

5.1.7.2 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 and servlet filters it is easy to register these web application elements without tracking the Http 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.name`	String	The value of this service property refers to a Http Context service. If this property is missing ~~or empty~~, the default context is used. If no context with the name exists or if the context is registered by another bundle and does not have the `osgi.http.whiteboard.context.shared` property set to `true`, the ~~servlet cannot be~~

		registeredwhiteboard service is ignored.
osgi.http.whiteboard.service.target	String	Services registered with this property are registered with an Http Service whose service registration properties match thisThe value of this service property is an LDAP filter expression which selects the Http 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.service.target` property containing a filter expression. A Http Service about to consumeprocess a servlet, servlet filter, listener, or resource must match that filter against his own service registration propertiesits runtime attributes. Only if the filter matches, the servlet, servlet filter, listener, or resource is used by the Http Service. For example a whiteboard service registered with the property

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

must only be used by an Http Service exposingwith the runtime attribute `osgi.http.implementation.name` property set tohaving the value `admin`.

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

If more than one Http 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 Services. If more than one Http 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 Services registered by different bundles but only with one Http Service from the same bundle.

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

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

5.2.2 Http Context for servlets, servlet filters, resources, and listeners

By default the whiteboard support is associating servlets, servlet filters, listeners, and resources with the default HttpContext of the targetted Http Service. Additional HttpContexts can be made available through the whiteboard support. In this case the HttpContext service must specify the `osgi.http.whiteboard.context.name` service property. This name can be used as a referenced by a servlet, servlet filter, listener, or resource services. If the HttpContext should be used by services from a different bundle than the bundle which registered the HttpContext service, the HttpContext must set the `osgi.http.whiteboard.context.shared` property to the bBoolean value `true`.

If there areis more than onemultiple, usable HttpContext services registeringed itself with the same context name, the Http Service implementation mustis usinguse the HttpContext with the highest service ranking. An HttpContext service is usable by a servlet, servlet filter, listener or resource if it is registered by the same bundle

or is shared. This might lead to re-binding ~~whiteboard services~~the servlet, servlet filter, listener or resource if a new usable HttpContext with a higher service ranking arrives or the current used HttpContext is unregistered.

BJH Not sure what you mean by rebinding here? Once the servlet, servlet filter, listener or resource is "bound" to a ServletContext backed by an HttpContext, how to you rebind the ServletContext to a newer HttpContext?

CZ: I've added above text, hope this clarifies it

~~If not specified further a servlet, servlet filter, listener, or resource using the whiteboard registration is associated with the default HttpContext. The service can be registered with a specific `HttpContext` by using the `osgi.http.whiteboard.context.name` service property.~~

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

Property	Type	Description
`osgi.http.whiteboard.context.name`	String+ or String	For `HttpContext` services this property is required and identifies the service when referred to by a servlet or servlet filter service. Http Context services without this property are ignored. For an HttpContext the type of this property is String+, as the context can be associated with more than one name. For other whiteboard service referencing a named HttpContext, the type of this property is String.
`osgi.http.whiteboard.context.shared`	Boolean	Whether a Http Context service may be used by servlet, listener, resource, or servlet filter services registered by other bundles. By default Http Context services can only be used by servlet, listener, resource, or servlet filter services registered by the same bundle.

5.2.3 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 HttpContext, 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 HttpContext.

~~If~~If a servlet is used by an HttpService implementation, the init() method of the servlet will be called. Once the servlet is no longer be used by the HttpService implementation the destroy() method will be called.

As servlet services might come and go as well as HttpContext services might come and go, the whiteboard service registration can be very dynamic. Therefore servlet services might transition between used by a HttpService implementation to not being used and back to be used. The servlet service should either be implemented in a reentrant way and be prepared that after a call of destroy() a new initialization through init() might follow. The recommended alternative is to use the prototype scope to implement and register servlets instead.

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.

5.2.4 Servlet Filter Registration

Servlet filters have been introduced into the Servlet API specification in Version 2.3 and thus far support for them ha~~ve~~s been ~~missing from~~absent in the Http Service specification. This update 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. A servlet filter can also reference servlets by name using the `osgi.http.whiteboard.filter.servlet` property.

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 HttpContext, 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 HttpService implementation, the init() method of the servlet filter will be called. Once the servlet filter is no longer be used by the HttpService implementation, the destroy() method will be called.

As servlet filter services might come and go as well as HttpContext services might come and go, the whiteboard service registration can be very dynamic. Therefore servlet filter services might transition between used by a HttpService implementation to not being used and back to be used. The servlet filter service should either be implemented in a reentrant way and be prepared that after a call of destroy() a new initialization through init() might follow. The recommended alternative is to use the prototype scope when registering and implementing a servlet filter.

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.
`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.

5.2.5 Resources

To register resources through the whiteboard an instance of the `org.osgi.service.http.ResourceServlet` servlet is registered as a regular servlet with the additional `osgi.http.whiteboard.resource.prefix` servlet registration property. The `osgi.http.whiteboard.~~resource~~servlet.pattern` property must ~~be a single value prefix pattern~~also be specified. ~~The `pattern` property is used as the alias and the `prefix` property as the name for the `registerResources` call.~~

Property	Type	Description
`osgi.http.whiteboard.resource.prefix`	String	~~Registers a mapping from the prefix pattern defined in the single valued `osgi.http.whiteboard.resource.pattern` to resources found at the given prefix.~~This prefix is used to map a requested resource to the bundle's entries. If this service property is not specified, a prefix of the empty string is used.

Example using DS:

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

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

```
    "osgi.http.whiteboard.context.name=resource-context"})
### BJH This wont work since ResourceServlet is a final class. So how do we register resource servlets in DS? Do we really need a ResourceServlet special (and otherwise unused) class? Can the presence of the resource.prefix service property be enough to mark the servlet service as a resource servlet?
### CZ We can get rid of ResourceServlet if we go back to have a separate property for the pattern like osgi.hhtp.whiteboard.resource.pattern . In this case a servlet registration can easily be distinguished from a resource registration.
public class MyResource extends ResourceServlet {
   ...
}
```

5.2.6 Event Listeners

Event listeners register themselves under the interface(s) they are implementing~~, s~~. This specification support~~sed are~~:

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

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

The ~~h~~Http ~~s~~Service implementation gets the listeners from the service registry as soon as the associated ~~h~~Http ~~c~~Context is established and releases them when the ~~c~~Http Context is not available any more or the listener is unregistered.

5.2.6.1 ServletContextListener and ServletContextAttributeListener

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

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

will always 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 Service decide to support dynamic registration through the servlet context, they ~~should~~ should require a proprietary opt-in mechanism like a manifest header or require capability.

5.2.7 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 ~~configured~~specified on a servlet service.~~, t~~ The property values can be an ~~http~~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.

5.3 Provided Capability

The Http Service implementation bundle ~~implementing whiteboard support for servlets, servlet filters, listeners, resources, and Http Contexts has to~~must provide the ~~following~~ `osgi.whiteboard` capability: for "osgi.http". For example:

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

~~Such a bundle~~The Http Service implementation must provide support for all whiteboard service types as outlined in this specification.

5.3.1.1 osgi.whiteboard Namespace

The whiteboard pattern leverages the OSGi service registry as a registry for objects. In the context of ~~the~~ Http Service, servlets can be registered ~~directly with the Http Service~~as services and the Http Service implementation ~~manages the usage of~~uses these ~~registrations~~services to interact with the servlets. ~~Applying the whiteboard pattern the services are registered with the OSGi service registry instead.~~

A ***Whiteboard*** *Services Consumer* is a ~~service~~bundle that ~~uses~~monitors the life cycle events ~~from o~~of ~~specific~~her services to use their functionality when the specific service~~s~~ ~~is~~are active. It can use metadata (service ~~registration~~ properties) to control its functionality. ***Whiteboard*** *Services* ***Providers***, register~~ing~~ 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 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 ~~specification~~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 ~~OSGi~~ Http 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 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)))"
```

6 Data Transfer Objects

This ~~chapter~~RFC defines an API to retrieve administrative information from the Http Service implementation. The HttpServiceRuntime service is introduced and can be called to obtain various ~~The~~ DTOs~~are accessed through the Http Service interface:~~

~~ServletDTO[] getServlets();~~
~~Map<String, String> getResources();~~
~~FilterDTO[] getFilters();~~
~~Map<String, String> getErrorLocations(); .~~

See the JavaDoc for details.

7 Javadoc

OSGi Javadoc

16.01.14 19:04

Package Summary		Page
org.osgi.service.http	Http Service Package Version 1.3.	24
org.osgi.service.http.runtime	Http Service Runtime Package Version 1.3.	41

Package org.osgi.service.http

```
@org.osgi.annotation.versioning.Version(value="1.3")
```

Http Service Package Version 1.3.

See:
Description

Interface Summary		Page
HttpContext	Context for HTTP Requests.	31
HttpService	**Deprecated.** *As of 1.3.*	34

Class Summary		Page
HttpConstants	Defines standard names for Http Service constants.	25
ResourceServlet	Bundle resource servlet.	39

Exception Summary		Page
NamespaceException	**Deprecated.** *As of 1.3.*	37

Package org.osgi.service.http Description

Http Service Package Version 1.3.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.http; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.http; version="[1.3,1.4)"
```

Class HttpConstants

org.osgi.service.http

```
java.lang.Object
  └ org.osgi.service.http.HttpConstants
```

```
final public class HttpConstants
extends Object
```

Defines standard names for Http Service constants.

Since:
1.3

Field Summary		Page
static String	**DISPATCHER_ASYNC** Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied in the async context.	29
static String	**DISPATCHER_ERROR** Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied when an error page is called.	29
static String	**DISPATCHER_FORWARD** Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to forward calls to the dispatcher.	29
static String	**DISPATCHER_INCLUDE** Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to include calls to the dispatcher.	29
static String	**DISPATCHER_REQUEST** Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to client requests.	29
static String	**HTTP_SERVICE_ENDPOINT_ATTRIBUTE** Http Service runtime attribute specifying the endpoints upon which the Http Service runtime is listening.	30
static String	**HTTP_WHITEBOARD_CONTEXT_NAME** Service property specifying the name of an HttpContext service.	26
static String	**HTTP_WHITEBOARD_CONTEXT_SHARED** Service property specifying whether an HttpContext service can be used by bundles other than the bundle which registered the Http Context service.	26
static String	**HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED** Service property specifying whether a Filter service supports asynchronous processing.	28
static String	**HTTP_WHITEBOARD_FILTER_DISPATCHER** Service property specifying the dispatcher handling of a Filter.	28
static String	**HTTP_WHITEBOARD_FILTER_NAME** Service property specifying the servlet filter name of a Filter service.	27
static String	**HTTP_WHITEBOARD_FILTER_PATTERN** Service property specifying the request mappings for a Filter service.	28
static String	**HTTP_WHITEBOARD_FILTER_SERVLET** Service property specifying the servlet names for a Filter service.	28
static String	**HTTP_WHITEBOARD_RESOURCE_PREFIX** Service property specifying the resource entry prefix for a ResourceServlet servlet service.	30
static String	**HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED** Service property specifying whether a Servlet service supports asynchronous processing.	27

static String	**HTTP_WHITEBOARD_SERVLET_ERROR_PAGE** Service property specifying whether a `Servlet` service acts as an error page.	27
static String	**HTTP_WHITEBOARD_SERVLET_NAME** Service property specifying the servlet name of a `Servlet` service.	26
static String	**HTTP_WHITEBOARD_SERVLET_PATTERN** Service property specifying the request mappings for a `Servlet` service.	27
static String	**HTTP_WHITEBOARD_TARGET** Service property specifying the target filter to select the Http Service runtime to process the service.	30

Field Detail

HTTP_WHITEBOARD_CONTEXT_NAME

```
public static final String HTTP_WHITEBOARD_CONTEXT_NAME = "osgi.http.whiteboard.context.name"
```

Service property specifying the name of an `HttpContext` service.

For `HttpContext` services, this service property must be specified. Http Context services without this service property must be ignored.

For servlet, listener, servlet filter, or resource servlet services, this service property refers to the name of the associated Http Context service. If this service property is not specified, then the default Http Conext must be used. If there is no Http Context service matching the specified name or the matching Http Context service is registered by another bundle but does not have the `HTTP_WHITEBOARD_CONTEXT_SHARED` service property set to true, the servlet, listener, servlet filter, or resource servlet service must be ignored.

For `HttpContext` services, the value of this service property must be of type `String`, `String[]`, or `Collection<String>`. For servlet, listener, servlet filter, or resource servlet services, the value of this service property must be of type `String`

See Also:
`HTTP_WHITEBOARD_CONTEXT_SHARED`

HTTP_WHITEBOARD_CONTEXT_SHARED

```
public static final String HTTP_WHITEBOARD_CONTEXT_SHARED = "osgi.http.whiteboard.context.shared"
```

Service property specifying whether an `HttpContext` service can be used by bundles other than the bundle which registered the Http Context service.

By default Http Context services can only be used by the bundle which registered the Http Context service.

The value of this service property must be of type `Boolean`.

See Also:
`HTTP_WHITEBOARD_CONTEXT_NAME`

HTTP_WHITEBOARD_SERVLET_NAME

```
public static final String HTTP_WHITEBOARD_SERVLET_NAME = "osgi.http.whiteboard.servlet.name"
```

Service property specifying the servlet name of a `Servlet` service.

This name is used as the value for the `ServletConfig.getServletName()` method. If this service property is not specified, the fully qualified name of the service object's class is used as the servlet name. Filter

services may refer to servlets by this name in their HTTP_WHITEBOARD_FILTER_SERVLET service property to apply the filter to the servlet.

Servlet names must be unique among all servlet services associated with an HttpContext. If multiple servlet services associated with the same HttpContext have the same servlet name, then all but the highest ranked servlet service must be ignored.

The value of this service property must be of type String.

HTTP_WHITEBOARD_SERVLET_PATTERN

```
public static final String HTTP_WHITEBOARD_SERVLET_PATTERN = "osgi.http.whiteboard.servlet.pattern"
```

Service property specifying the request mappings for a Servlet service.

The specified patterns are used to determine whether a request should be mapped to the servlet. Servlet services without this service property or HTTP_WHITEBOARD_SERVLET_ERROR_PAGE must be ignored.

The value of this service property must be of type String, String[], or Collection<String>.

See Also:
"Java Servlet Specification Version 3.0, Section 12.2 Specification of Mappings"

HTTP_WHITEBOARD_SERVLET_ERROR_PAGE

```
public static final String HTTP_WHITEBOARD_SERVLET_ERROR_PAGE = "osgi.http.whiteboard.servlet.errorPage"
```

Service property specifying whether a Servlet service acts as an error page.

The service property values may be the name of a fully qualified exception class or a three digit HTTP status code. Any value that is not a three digit number is considered to be the name of a fully qualified exception class.

The value of this service property must be of type String, String[], or Collection<String>.

HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED

```
public static final String HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED = "osgi.http.whiteboard.servlet.asyncSupported"
```

Service property specifying whether a Servlet service supports asynchronous processing.

By default Servlet services do not support asynchronous processing.

The value of this service property must be of type Boolean.

See Also:
"Java Servlet Specification Version 3.0, Section 2.3.3.3 Asynchronous Processing"

HTTP_WHITEBOARD_FILTER_NAME

```
public static final String HTTP_WHITEBOARD_FILTER_NAME = "osgi.http.whiteboard.filter.name"
```

Service property specifying the servlet filter name of a Filter service.

This name is used as the value for the `FilterConfig.getFilterName()` method. If this service property is not specified, the fully qualified name of the service object's class is used as the servlet filter name.

Servlet filter names must be unique among all servlet filter services associated with an `HttpContext`. If multiple servlet filter services associated with the same HttpContext have the same servlet filter name, then all but the highest ranked servlet filter service must be ignored.

The value of this service property must be of type `String`.

HTTP_WHITEBOARD_FILTER_PATTERN

```
public static final String HTTP_WHITEBOARD_FILTER_PATTERN = "osgi.http.whiteboard.filter.pattern"
```

Service property specifying the request mappings for a `Filter` service.

The specified patterns are used to determine whether a request should be mapped to the servlet filter. Filter services without this service property or the `HTTP_WHITEBOARD_FILTER_SERVLET` service property must be ignored.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`.

See Also:
"Java Servlet Specification Version 3.0, Section 12.2 Specification of Mappings"

HTTP_WHITEBOARD_FILTER_SERVLET

```
public static final String HTTP_WHITEBOARD_FILTER_SERVLET = "osgi.http.whiteboard.filter.servlet"
```

Service property specifying the `servlet names` for a `Filter` service.

The specified names are used to determine the servlets whose requests should be mapped to the servlet filter. Filter services without this service property or the `HTTP_WHITEBOARD_FILTER_PATTERN` service property must be ignored.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`.

HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED

```
public static final String HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED = "osgi.http.whiteboard.filter.asyncSupported"
```

Service property specifying whether a `Filter` service supports asynchronous processing.

By default Filters services do not support asynchronous processing.

The value of this service property must be of type `Boolean`.

See Also:
"Java Servlet Specification Version 3.0, Section 2.3.3.3 Asynchronous Processing"

HTTP_WHITEBOARD_FILTER_DISPATCHER

```
public static final String HTTP_WHITEBOARD_FILTER_DISPATCHER = "osgi.http.whiteboard.filter.dispatcher"
```

Service property specifying the dispatcher handling of a `Filter`.

By default Filters services are associated with client requests only (see value DISPATCHER_REQUEST.

The value of this service property must be of type `String`, `String[]`, or `Collection<String>`. Allowed values are DISPATCHER_ASYNC, DISPATCHER_ERROR, DISPATCHER_FORWARD, DISPATCHER_INCLUDE, DISPATCHER_REQUEST.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

DISPATCHER_REQUEST

```
public static final String DISPATCHER_REQUEST = "REQUEST"
```

Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to client requests.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

DISPATCHER_INCLUDE

```
public static final String DISPATCHER_INCLUDE = "INCLUDE"
```

Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to include calls to the dispatcher.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

DISPATCHER_FORWARD

```
public static final String DISPATCHER_FORWARD = "FORWARD"
```

Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied to forward calls to the dispatcher.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

DISPATCHER_ASYNC

```
public static final String DISPATCHER_ASYNC = "ASYNC"
```

Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied in the async context.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

DISPATCHER_ERROR

```
public static final String DISPATCHER_ERROR = "ERROR"
```

Possible value for the HTTP_WHITEBOARD_FILTER_DISPATCHER property indicating the filter is applied when an error page is called.

See Also:
"Java Servlet Specification Version 3.0, Section 6.2.5 Filters and the RequestDispatcher"

HTTP_WHITEBOARD_RESOURCE_PREFIX

```
public static final String HTTP_WHITEBOARD_RESOURCE_PREFIX = "osgi.http.whiteboard.resource.prefix"
```

Service property specifying the resource entry prefix for a ResourceServlet servlet service.

This prefix is used to map a requested resource to the bundle's entries. If this service property is not specified, a prefix of the empty string is used.

The value of this service property must be of type `String`.

HTTP_WHITEBOARD_TARGET

```
public static final String HTTP_WHITEBOARD_TARGET = "osgi.http.whiteboard.target"
```

Service property specifying the target filter to select the Http Service runtime to process the service.

An Http Service implementation can define any number of attributes which can be referenced by the target filter. The attributes should always include the osgi.http.endpoint attribute if the endpoint information is known.

If this service property is not specified, then all Http Service runtimes can process the service.

The value of this service property must be of type `String` and be a valid `filter string`.

HTTP_SERVICE_ENDPOINT_ATTRIBUTE

```
public static final String HTTP_SERVICE_ENDPOINT_ATTRIBUTE = "osgi.http.endpoint"
```

Http Service runtime attribute specifying the endpoints upon which the Http Service runtime is listening.

An endpoint value is a URL to which the Http Service runtime is listening. For example, `http://192.168.1.10:8080/`. The relevant information contained in the URL is the scheme, IP Address of the bound interface, bound port, and the (optional) context path in a Servlet API servlet container for the Http Service runtime. An Http Service Runtime can be listening on multiple endpoints.

The value of this attribute must be of type `String`, `String[]`, or `Collection<String>`.

Interface HttpContext

org.osgi.service.http

```
@org.osgi.annotation.versioning.ConsumerType
public interface HttpContext
```

Context for HTTP Requests.

This service defines methods that the Http Service may call to get information for a request.

Servlets may be associated with an HttpContext service. Servlets that are associated using the same HttpContext object will share the same ServletContext object.

If no HttpContext service is associated, a default HttpContext is used. The behavior of the methods on the default HttpContext is defined as follows:

1. getMimeType - Does not define any customized MIME types for the Content-Type header in the response, and always returns null.
2. handleSecurity - Performs implementation-defined authentication on the request.
3. getResource - Assumes the named resource is in the bundle of the servlet service. This method calls the servlet bundle's Bundle.getResource method, and returns the appropriate URL to access the resource. On a Java runtime environment that supports permissions, the Http Service needs to be granted org.osgi.framework.AdminPermission[*,RESOURCE].

See Also:
HttpConstants.HTTP_WHITEBOARD_CONTEXT_NAME, HttpConstants.HTTP_WHITEBOARD_CONTEXT_SHARED

ThreadSafe

Field Summary		Page
String	**AUTHENTICATION_TYPE** HttpServletRequest attribute specifying the scheme used in authentication.	32
String	**AUTHORIZATION** HttpServletRequest attribute specifying the Authorization object obtained from the org.osgi.service.useradmin.UserAdmin service.	32
String	**REMOTE_USER** HttpServletRequest attribute specifying the name of the authenticated user.	31

Method Summary		Page
String	**getMimeType**(String name) Maps a name to a MIME type.	33
URL	**getResource**(String name) Maps a resource name to a URL.	33
boolean	**handleSecurity**(HttpServletRequest request, HttpServletResponse response) Handles security for the specified request.	32

Field Detail

REMOTE_USER

```
public static final String REMOTE_USER = "org.osgi.service.http.authentication.remote.user"
```

HttpServletRequest attribute specifying the name of the authenticated user. The value of the attribute can be retrieved by HttpServletRequest.getRemoteUser. This attribute name is org.osgi.service.http.authentication.remote.user.

Since:
1.1

AUTHENTICATION_TYPE

```
public static final String AUTHENTICATION_TYPE = "org.osgi.service.http.authentication.type"
```

`HttpServletRequest` attribute specifying the scheme used in authentication. The value of the attribute can be retrieved by `HttpServletRequest.getAuthType`. This attribute name is `org.osgi.service.http.authentication.type`.

Since:
1.1

AUTHORIZATION

```
public static final String AUTHORIZATION = "org.osgi.service.useradmin.authorization"
```

`HttpServletRequest` attribute specifying the `Authorization` object obtained from the `org.osgi.service.useradmin.UserAdmin` service. The value of the attribute can be retrieved by `HttpServletRequest.getAttribute(HttpContext.AUTHORIZATION)`. This attribute name is `org.osgi.service.useradmin.authorization`.

Since:
1.1

Method Detail

handleSecurity

```
boolean handleSecurity(HttpServletRequest request,
                       HttpServletResponse response)
                throws IOException
```

Handles security for the specified request.

The Http Service calls this method prior to servicing the specified request. This method controls whether the request is processed in the normal manner or an error is returned.

If the request requires authentication and the Authorization header in the request is missing or not acceptable, then this method should set the WWW-Authenticate header in the response object, set the status in the response object to Unauthorized(401) and return `false`. See also RFC 2617: *HTTP Authentication: Basic and Digest Access Authentication* (available at http://www.ietf.org/rfc/rfc2617.txt).

If the request requires a secure connection and the `getScheme` method in the request does not return 'https' or some other acceptable secure protocol, then this method should set the status in the response object to Forbidden(403) and return `false`.

When this method returns `false`, the Http Service will send the response back to the client, thereby completing the request. When this method returns `true`, the Http Service will proceed with servicing the request.

If the specified request has been authenticated, this method must set the `AUTHENTICATION_TYPE` request attribute to the type of authentication used, and the `REMOTE_USER` request attribute to the remote user (request attributes are set using the `setAttribute` method on the request). If this method does not perform any authentication, it must not set these attributes.

If the authenticated user is also authorized to access certain resources, this method must set the `AUTHORIZATION` request attribute to the `Authorization` object obtained from the `org.osgi.service.useradmin.UserAdmin` service.

The servlet responsible for servicing the specified request determines the authentication type and remote user by calling the `getAuthType` and `getRemoteUser` methods, respectively, on the request.

Parameters:
`request` - The HTTP request.
`response` - The HTTP response.
Returns:
`true` if the request should be serviced, `false` if the request should not be serviced and Http Service will send the response back to the client.
Throws:
`IOException` - may be thrown by this method. If this occurs, the Http Service will terminate the request and close the socket.

---

getResource

```
URL getResource(String name)
```

Maps a resource name to a URL.

Called by the Http Service to map the specified resource name to a URL. For servlets, Http Service will call this method to support the `ServletContext` methods `getResource` and `getResourceAsStream`. For `ResourceServlet` servlets, Http Service will call this method to locate the named resource.

The context can control from where resources come. For example, the resource can be mapped to a file in the bundle's persistent storage area via `bundleContext.getDataFile(name).toURL()` or to a resource in the context's bundle via `getClass().getResource(name)`

Parameters:
`name` - The name of the requested resource.
Returns:
A URL that Http Service can use to read the resource or `null` if the resource does not exist.

---

getMimeType

```
String getMimeType(String name)
```

Maps a name to a MIME type.

Called by the Http Service to determine the MIME type for the specified name. For servlets, the Http Service will call this method to support the `ServletContext` method `getMimeType`. For `ResourceServlet` servlets, the Http Service will call this method to determine the MIME type for the `Content-Type` header in the response.

Parameters:
`name` - The name for which to determine the MIME type.
Returns:
The MIME type (e.g. text/html) of the specified name or `null` to indicate that the Http Service should determine the MIME type itself.

Interface HttpService

org.osgi.service.http

```
@org.osgi.annotation.versioning.ProviderType
public interface HttpService
```

Deprecated.

The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service. A bundle may later unregister its resources or servlets.

See Also:
HttpContext
ThreadSafe

Method Summary		Page
HttpContext	**createDefaultHttpContext**() Creates a default HttpContext for registering servlets or resources with the HttpService, a new HttpContext object is created each time this method is called.	36
void	**registerResources**(String alias, String name, HttpContext context) Registers resources into the URI namespace.	35
void	**registerServlet**(String alias, Servlet servlet, Dictionary<String,String> initparams, HttpContext context) Registers a servlet into the URI namespace.	34
void	**unregister**(String alias) Unregisters a previous registration done by registerServlet or registerResources methods.	35

Method Detail

registerServlet

```
void registerServlet(String alias,
                     Servlet servlet,
                     Dictionary<String,String> initparams,
                     HttpContext context)
              throws ServletException,
                     NamespaceException
```

Registers a servlet into the URI namespace.

The alias is the name in the URI namespace of the Http Service at which the registration will be mapped.

An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form "/" is used to denote the root alias. See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

The Http Service will call the servlet's init method before returning.

```
httpService.registerServlet("/myservlet", servlet, initparams, context);
```

Servlets registered with the same HttpContext object will share the same ServletContext. The Http Service will call the context argument to support the ServletContext methods getResource,getResourceAsStream and getMimeType, and to handle security for requests. If the context argument is null, a default HttpContext object is used (see createDefaultHttpContext()).

Parameters:
alias - name in the URI namespace at which the servlet is registered

servlet - the servlet object to register
initparams - initialization arguments for the servlet or null if there are none. This argument is used by the servlet's ServletConfig object.
context - the HttpContext object for the registered servlet, or null if a default HttpContext is to be created and used.

Throws:

ServletException - if the servlet's init method throws an exception, or the given servlet object has already been registered at a different alias.
NamespaceException - if the registration fails because the alias is already in use.
IllegalArgumentException - if any of the arguments are invalid

---

registerResources

```
void registerResources(String alias,
                       String name,
                       HttpContext context)
                throws NamespaceException
```

Registers resources into the URI namespace.

The alias is the name in the URI namespace of the Http Service at which the registration will be mapped. An alias must begin with slash ('/') and must not end with slash ('/'), with the exception that an alias of the form "/" is used to denote the root alias. The name parameter must also not end with slash ('/') with the exception that a name of the form "/" is used to denote the root of the bundle. See the specification text for details on how HTTP requests are mapped to servlet and resource registrations.

For example, suppose the resource name /tmp is registered to the alias /files. A request for /files/foo.txt will map to the resource name /tmp/foo.txt.

```
httpservice.registerResources("/files", "/tmp", context);
```

The Http Service will call the HttpContext argument to map resource names to URLs and MIME types and to handle security for requests. If the HttpContext argument is null, a default HttpContext is used (see createDefaultHttpContext()).

Parameters:

alias - name in the URI namespace at which the resources are registered
name - the base name of the resources that will be registered
context - the HttpContext object for the registered resources, or null if a default HttpContext is to be created and used.

Throws:

NamespaceException - if the registration fails because the alias is already in use.
IllegalArgumentException - if any of the parameters are invalid

---

unregister

```
void unregister(String alias)
```

Unregisters a previous registration done by registerServlet or registerResources methods.

After this call, the registered alias in the URI name-space will no longer be available. If the registration was for a servlet, the Http Service must call the destroy method of the servlet before returning.

If the bundle which performed the registration is stopped or otherwise "unget"s the Http Service without calling unregister(String) then Http Service must automatically unregister the registration. However, if the registration was for a servlet, the destroy method of the servlet will not be called in this case since the bundle may be stopped. unregister(String) must be explicitly called to cause the destroy method of the servlet to be called. This can be done in the BundleActivator.stop method of the bundle registering the servlet.

Parameters:

alias - name in the URI name-space of the registration to unregister

Throws:

`IllegalArgumentException` - if there is no registration for the alias or the calling bundle was not the bundle which registered the alias.

---

createDefaultHttpContext

`HttpContext` **`createDefaultHttpContext`**`()`

Creates a default `HttpContext` for registering servlets or resources with the HttpService, a new `HttpContext` object is created each time this method is called.

The behavior of the methods on the default `HttpContext` is defined as follows:

- `getMimeType` - Does not define any customized MIME types for the Content-Type header in the response, and always returns `null`.
- `handleSecurity` - Performs implementation-defined authentication on the request.
- `getResource` - Assumes the named resource is in the context bundle; this method calls the context bundle's `Bundle.getResource` method, and returns the appropriate URL to access the resource. On a Java runtime environment that supports permissions, the Http Service needs to be granted `org.osgi.framework.AdminPermission[*,RESOURCE]`.

Returns:

a default `HttpContext` object.

Since:

1.1

Class NamespaceException

org.osgi.service.http

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └org.osgi.service.http.NamespaceException
```

All Implemented Interfaces:
Serializable

```
public class NamespaceException
extends Exception
```

Deprecated.

A NamespaceException is thrown to indicate an error with the caller's request to register a servlet or resources into the URI namespace of the Http Service. This exception indicates that the requested alias already is in use.

Constructor Summary	Page
`NamespaceException(String message)` Construct a `NamespaceException` object with a detail message.	37
`NamespaceException(String message, Throwable cause)` Construct a `NamespaceException` object with a detail message and a nested exception.	37

Method Summary		Page
Throwable	`getCause()` Returns the cause of this exception or `null` if no cause was set.	38
Throwable	`getException()` Returns the nested exception.	38
Throwable	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	38

Constructor Detail

NamespaceException

```
public NamespaceException(String message)
```

Construct a `NamespaceException` object with a detail message.

Parameters:
`message` - the detail message

NamespaceException

```
public NamespaceException(String message,
                          Throwable cause)
```

Construct a `NamespaceException` object with a detail message and a nested exception.

Parameters:
`message` - The detail message.
`cause` - The nested exception.

Method Detail

getException

```
public Throwable getException()
```

Returns the nested exception.

This method predates the general purpose exception chaining mechanism. The `getCause()` method is now the preferred means of obtaining this information.

Returns:
The result of calling `getCause()`.

---

getCause

```
public Throwable getCause()
```

Returns the cause of this exception or `null` if no cause was set.

Overrides:
`getCause` in class `Throwable`
Returns:
The cause of this exception or `null` if no cause was set.
Since:
1.2

---

initCause

```
public Throwable initCause(Throwable cause)
```

Initializes the cause of this exception to the specified value.

Overrides:
`initCause` in class `Throwable`
Parameters:
`cause` - The cause of this exception.
Returns:
This exception.
Throws:
`IllegalArgumentException` - If the specified cause is this exception.
`IllegalStateException` - If the cause of this exception has already been set.
Since:
1.2

Class ResourceServlet

org.osgi.service.http

```
java.lang.Object
  └javax.servlet.GenericServlet
      └org.osgi.service.http.ResourceServlet
```

All Implemented Interfaces:
Serializable, Servlet, ServletConfig

```
final public class ResourceServlet
extends GenericServlet
```

Bundle resource servlet.

The `ResourceServlet` is a marker servlet which can be used to serve bundle resources. A `ResourceServlet` object is registered as a `javax.servlet.Servlet` service along with the `HttpConstants.HTTP_WHITEBOARD_SERVLET_NAME`, `HttpConstants.HTTP_WHITEBOARD_SERVLET_PATTERN`, `HttpConstants.HTTP_WHITEBOARD_RESOURCE_PREFIX`, and `HttpConstants.HTTP_WHITEBOARD_CONTEXT_NAME` service properties.

Since:
1.3
ThreadSafe

Constructor Summary	Page
ResourceServlet() ResourceServlet constructor.	39

Method Summary		Page
void	**service**(ServletRequest request, ServletResponse response) ResourceServlet service method.	39

Constructor Detail

ResourceServlet

```
public ResourceServlet()
```

ResourceServlet constructor.

This method implementation is empty.

Method Detail

service

```
public void service(ServletRequest request,
                    ServletResponse response)
```

ResourceServlet service method.

This method implementation is empty. This method is not called or used by Http Service.

Specified by:
`service` in interface `Servlet`
Overrides:
`service` in class `GenericServlet`
Parameters:
`request` - The HTTP request.
`response` - The HTTP response.

Package org.osgi.service.http.runtime

```
@org.osgi.annotation.versioning.Version(value="1.3")
```

Http Service Runtime Package Version 1.3.

See:
Description

Interface Summary		Page
HttpServiceRuntime	The `HttpServiceRuntime` service represents the runtime information of an Http Service implementation.	*44*

Class Summary		Page
FilterDTO	Represents a servlet `Filter` service used by the Http Service runtime.	*42*
ListenerDTO	Represents a listener service used by the Http Service runtime.	*47*
ResourceServletDTO	Represents a `ResourceServlet` servlet service used by the Http Service runtime.	*48*
ServletContextDTO	Represents a `ServletContext` created for registered servlets and servlet filters backed by a `HttpContext` service.	*50*
ServletDTO	Represents a `Servlet` service used by the Http Service runtime.	*52*

Package org.osgi.service.http.runtime Description

Http Service Runtime Package Version 1.3.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.http.runtime; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.http.runtime; version="[1.3,1.4)"
```

Class FilterDTO

org.osgi.service.http.runtime

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.http.runtime.FilterDTO
```

```
public class FilterDTO
extends org.osgi.dto.DTO
```

Represents a servlet `Filter` service used by the Http Service runtime.

Since:
1.3
NotThreadSafe

Field Summary		Page
boolean	**asyncSupported** Specifies whether the servlet filter supports asynchronous processing.	43
String[]	**dispatcher** The dispatcher associations for the servlet filter.	43
String	**name** The name of the servlet filter.	42
String[]	**patterns** The request mappings for the servlet filter.	42
ServletContextDTO	**servletContext** The ServletContextDTO representing the `ServletContext` for the servlet filter.	43
String[]	**servletNames** The servlet names for the servlet filter.	43

Constructor Summary	Page
FilterDTO()	43

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the servlet filter.

See Also:
HttpConstants.HTTP_WHITEBOARD_FILTER_NAME

---

patterns

```
public String[] patterns
```

The request mappings for the servlet filter.

The specified patterns are used to determine whether a request should be mapped to the servlet filter.

See Also:
HttpConstants.HTTP_WHITEBOARD_FILTER_PATTERN

---

servletNames

```
public String[] servletNames
```

The servlet names for the servlet filter.

The specified names are used to determine the servlets whose requests should be mapped to the servlet filter.

See Also:
HttpConstants.HTTP_WHITEBOARD_FILTER_NAME

---

asyncSupported

```
public boolean asyncSupported
```

Specifies whether the servlet filter supports asynchronous processing.

See Also:
HttpConstants.HTTP_WHITEBOARD_FILTER_ASYNC_SUPPORTED

---

dispatcher

```
public String[] dispatcher
```

The dispatcher associations for the servlet filter.

The specified names are used to determine in what occasions the servlet filter is called

See Also:
HttpConstants.HTTP_WHITEBOARD_FILTER_DISPATCHER

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletContextDTO representing the `ServletContext` for the servlet filter.

Constructor Detail

FilterDTO

```
public FilterDTO()
```

Interface HttpServiceRuntime

org.osgi.service.http.runtime

```
@org.osgi.annotation.versioning.ProviderType
public interface HttpServiceRuntime
```

The `HttpServiceRuntime` service represents the runtime information of an Http Service implementation.

It provides access to the servlet, listener, servlet filter, or resource servlet services used by the Http Service runtime.

Since:
1.3
ThreadSafe

Method Summary		Page
Map<String,Object>	**getAttributes**() Returns the attributes of this Http Service runtime.	44
Map<String,String>	**getErrorLocations**() TODO ### BJH I don't understand the mapping here and how it can be used by anyone.	45
FilterDTO[]	**getFilterDTOs**() Returns the representations of the servlet `Filter` services used by this Http Service runtime.	45
ListenerDTO[]	**getListenerDTOs**() Returns the representations of the listener services used by this Http Service runtime.	45
ResourceServletDTO[]	**getResourceServletDTOs**() Returns the representations of the `ResourceServlet` servlet services used by this Http Service runtime.	45
ServletContextDTO[]	**getServletContextDTOs**() Returns the representations of the `ServletContext` objects used by this Http Service runtime.	45
ServletDTO[]	**getServletDTOs**() Returns the representations of the `Servlet` services used by this Http Service runtime.	44

Method Detail

getAttributes

```
Map<String,Object> getAttributes()
```

Returns the attributes of this Http Service runtime.

The attributes must always include the `osgi.http.endpoint` attribute.

Returns:
The attributes of this Http Service runtime.

getServletDTOs

```
ServletDTO[] getServletDTOs()
```

Returns the representations of the `Servlet` services used by this Http Service runtime.

Returns:
The representations of the `Servlet` services used by this Http Service runtime. The returned array may be empty if this Http Service runtime is currently not using any `Servlet` services.

---

getResourceServletDTOs

`ResourceServletDTO[] getResourceServletDTOs()`

Returns the representations of the `ResourceServlet` servlet services used by this Http Service runtime.

Returns:
The representations of the `ResourceServlet` servlet services used by this Http Service runtime. The returned array may be empty if this Http Service runtime is currently not using any `ResourceServlet` servlet services.

---

getFilterDTOs

`FilterDTO[] getFilterDTOs()`

Returns the representations of the servlet `Filter` services used by this Http Service runtime.

Returns:
The representations of the servlet `Filter` services used by this Http Service runtime. The returned array may be empty if this Http Service runtime is currently not using any servlet `Filter` services.

---

getErrorLocations

`Map<String,String> getErrorLocations()`

TODO ### BJH I don't understand the mapping here and how it can be used by anyone. It seems what you want is an array of ErrorPageDTOs which contain the info in the `HttpConstants.HTTP_WHITEBOARD_SERVLET_ERROR_PAGE` service property and a pointer to a ServletDTO.

CZ I think getErrorPages() return a map with key String and value ServletDTO should do the trick.

Returns:
defined error location mappings or `null` if no error location mappings have been registered.

---

getServletContextDTOs

`ServletContextDTO[] getServletContextDTOs()`

Returns the representations of the `ServletContext` objects used by this Http Service runtime.

Returns:
The representations of the `ServletContext` objects used by this Http Service runtime. The returned array may be empty if this Http Service runtime is currently not using any `ServletContext` objects.

---

getListenerDTOs

`ListenerDTO[] getListenerDTOs()`

Returns the representations of the listener services used by this Http Service runtime.

Returns:

The representations of the listener services used by this Http Service runtime. The returned array may be empty if this Http Service runtime is currently not using any listener services.

Class ListenerDTO

org.osgi.service.http.runtime

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.http.runtime.ListenerDTO
```

```
public class ListenerDTO
extends org.osgi.dto.DTO
```

Represents a listener service used by the Http Service runtime.

Since:
1.3
NotThreadSafe

Field Summary		Page
ServletContextDTO	**servletContext** The ServletContextDTO representing the ServletContext for the listener.	47
String	**type** The fully qualified type name the listener.	47

Constructor Summary	Page
ListenerDTO()	47

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

type

```
public String type
```

The fully qualified type name the listener.

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletContextDTO representing the ServletContext for the listener.

Constructor Detail

ListenerDTO

```
public ListenerDTO()
```

Class ResourceServletDTO

org.osgi.service.http.runtime

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.http.runtime.ResourceServletDTO
```

```
public class ResourceServletDTO
extends org.osgi.dto.DTO
```

Represents a `ResourceServlet` servlet service used by the Http Service runtime.

Since:
1.3

NotThreadSafe

Field Summary		Page
String	**name** The name of the resource servlet.	48
String[]	**patterns** The request mappings for the resource servlet.	48
String	**prefix** The prefix of the resource servlet.	49
ServletContextDTO	**servletContext** The `ServletContextDTO` representing the `ServletContext` for the resource servlet.	49

Constructor Summary	Page
ResourceServletDTO()	49

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the resource servlet.

See Also:
`HttpConstants.HTTP_WHITEBOARD_SERVLET_NAME`

---

patterns

```
public String[] patterns
```

The request mappings for the resource servlet.

The specified patterns are used to determine whether a request should be mapped to the resource servlet.

See Also:
HttpConstants.HTTP_WHITEBOARD_SERVLET_PATTERN

---

prefix

```
public String prefix
```

The prefix of the resource servlet.

See Also:
HttpConstants.HTTP_WHITEBOARD_RESOURCE_PREFIX

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletContextDTO representing the ServletContext for the resource servlet.

Constructor Detail

ResourceServletDTO

```
public ResourceServletDTO()
```

Class ServletContextDTO

org.osgi.service.http.runtime

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.http.runtime.ServletContextDTO
```

```
public class ServletContextDTO
extends org.osgi.dto.DTO
```

Represents a `ServletContext` created for registered servlets and servlet filters backed by a `HttpContext` service.

Since:
1.3
NotThreadSafe

Field Summary		Page
`Map<String,Object>`	**attributes** The servlet context attributes.	51
`String`	**contextPath** The servlet context path.	50
`Map<String,String>`	**initParams** The servlet context initialization parameters.	51
`String`	**name** The name of the servlet context.	50

Constructor Summary	Page
ServletContextDTO()	51

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the servlet context.

This is the value returned by the `ServletContext.getServletContextName()` method.

---

contextPath

```
public String contextPath
```

The servlet context path. This is the value returned by the `ServletContext.getContextPath()` method.

---

initParams

```
public Map<String,String> initParams
```

The servlet context initialization parameters.

---

attributes

```
public Map<String,Object> attributes
```

The servlet context attributes.

The value type must be a numerical type, Boolean, String, DTO or an array of any of the former. Therefore this method will only return the attributes of the servlet context conforming to this constraint.

Constructor Detail

ServletContextDTO

```
public ServletContextDTO()
```

Class ServletDTO

org.osgi.service.http.runtime

```
java.lang.Object
  └org.osgi.dto.DTO
      └org.osgi.service.http.runtime.ServletDTO
```

```
public class ServletDTO
extends org.osgi.dto.DTO
```

Represents a `Servlet` service used by the Http Service runtime.

Since:
1.3
NotThreadSafe

Field Summary		Page
boolean	**asyncSupported** Specifies whether the servlet supports asynchronous processing.	53
String	**name** The name of the servlet.	52
String[]	**patterns** The request mappings for the servlet.	52
ServletContextDTO	**servletContext** The ServletContextDTO representing the ServletContext for the servlet.	53
String	**servletInfo** The information string from the servlet.	53

Constructor Summary	Page
ServletDTO()	53

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

name

```
public String name
```

The name of the servlet.

See Also:
HttpConstants.HTTP_WHITEBOARD_SERVLET_NAME

---

patterns

```
public String[] patterns
```

The request mappings for the servlet.

The specified patterns are used to determine whether a request should be mapped to the servlet.

See Also:
HttpConstants.HTTP_WHITEBOARD_SERVLET_PATTERN

---

servletInfo

```
public String servletInfo
```

The information string from the servlet.

This is the value returned by the `Servlet.getServletInfo()` method.

---

asyncSupported

```
public boolean asyncSupported
```

Specifies whether the servlet supports asynchronous processing.

See Also:
HttpConstants.HTTP_WHITEBOARD_SERVLET_ASYNC_SUPPORTED

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletContextDTO representing the `ServletContext` for the servlet.

Constructor Detail

ServletDTO

```
public ServletDTO()
```


8 Considered Alternatives

8.1 Servlet API Reference Version

This specification is based on Servlet API 3.0. Implementations though are free to be based on any prior or later Servlet API specification. The specification must still be available to implementations in embedded environments which are still mostly based on Java ME corresponding to Java 1.4.

Therefore the specification cannot mandate either Servlet API 2.5 whose specification requires Java 5 or Servlet API 3.0 whose specification requires Java 6 even though none of the API really requires the respective platforms.

8.2 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.

8.3 Web Application Events

8.3.1 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 Services sending events to the listener service.

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

8.3.2 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.

8.4 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.

8.5 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.

8.6 Deprecated HttpService

The complete HttpService interface is now deprecated. A new HttpServiceRuntime interface is introduced in the org.osgi.service.http.runtime package to obtain runtime information about the Http Service implementation in the form of DTOs.

New service properties that were defined by this RFC for the now deprecated HttpService are now available as runtime attributes from the HttpServiceRuntime service.

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 http contexts 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 http context registered by another bundle, the bundle registering the whiteboard service must be granted ServicePermission[org.osgi.service.http.HttpContext, GET].

Bundles that need to introspect the state of the Http 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