RFC 189 Http Service Updates

Draft

47 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

Revision	Date	Comments
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
Update	08/08/13	Updated with feedback from BJ (partially already mentioned at the Palo Alto F2F) : • Clean up requirements list • Several clarifications / rewordings Carsten Ziegeler, Adobe Systems Incorporated, cziegele@adobe.com

1 Introduction

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

- Http Service Specification based on Servlet API 2.1. Apart from being based an old Servlet API version and being silent about how more recent versions are supported the main problem with this specification is that a provider of servlets and resources has to grab the Http Service first before being able to register servlets and resources. There is no whiteboard pattern support.
- Web Applications Specification basically just defines how existing web applications may be enhanced with OSGi Manifest headers and deployed into the OSGi Framework as-is. This is fine for moving existing web applications with minimal changes into the OSGi framework.

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

2 Application Domain

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

3 Problem Description

3.1 Support for dated 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 define the relationship between the Http Service and Web Application specifications.~~

HS-2 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-3 The solution MUST extend the HttpService 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-4 The solution MUST extend the HttpService 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-5 The solution MUST extend the HttpService service API to support registration of Servlet API listeners.

HS-6 The solution MUST add support for error page configuration.

HS-7 The solution MUST define how registered Servlets and Filters are named.

HS-8 The solution MUST clarify ServletContext implementation in the HttpService for both standalone and bridged Http Service implementations.

HS-9 The solution MUST clarify the ServletContext scope of Servlet API listeners registered through the HttpService.

HS-10 ~~The solution MAY specify support for scripted request processing. For example supporting JSP with Tag Libraries.~~

HS-11 ~~The solution MAY define how HttpService instances can be dynamically configured.~~

HS-12 The solution MUST define service registration properties for the HttpService to reflect configuration of the service.

HS-13

HS-14 The solution MUST define whiteboard registration of servlet services with the HttpService.

HS-15 The solution MUST define whiteboard registration of filter services with the HttpService.

HS-16 The solution MUST define whiteboard registration of servlet listener services with the HttpService.

HS-17 The solution MUST define registration of OSGi HttpContext services used for Servlet and Filter registration.

HS-18 The solution MUST define how servlets, filters, and servlet listener services are matched with HttpService services for registration.

HS-19 The solution MUST define whiteboard registration ~~support registration~~ of static resources ~~according to the extender pattern~~with the HttpService.

HS-20 The solution MUST define whiteboard registration ~~support registration~~ of error pages ~~according to the extender pattern~~with the HttpService.

HS-21 ~~The solution MUST define a capability for the `osgi.c` `nder` namespace. Bundles providing resources and/or error pages can then require this capability.~~

HS-22 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, Filters, 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)
- 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, Filters, 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, Filter, and Resources:

- `ServletContext` objects used for Servlet and Filter initialization
- Http Sessions acquired by Servlets and 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
`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`
`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`
`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, 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. In the context of Http Service we don't have and support such listeners. All listeners supported are registered in the OSGi service registry and thus controlled by their respective registering bundles.

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 a service property `osgi.http.whiteboard.asyncSupported`.

5.1.5 Http Service

5.1.5.1 Service Registration Properties

The Http Service must expose the following information through its servlet registration properties:

`osgi.http.service.endpoints`	A String+ property 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 may not always be available to the Http Service implementation, particularly in a bridged implementation. In such cases these properties may be omitted from the service registration.

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 suggested exports and capabilities.

If the Servlet API is provided by another bundle, it is recommended to export the packages and provide the capabilities as listed.

5.1.7.1 Providing Serlvet API 3.0

```
Export-Package: javax.servlet, javax.servlet.http
Provide-Capability: osgi.contract; osgi.contract=java.servlet; version:Version=3;
 uses:="javax.servlet, javax.servlet.http", osgi.contract;
 osgi.contract=java.servlet; version:Version=2.5; uses:="javax.servlet,
 javax.servlet.http", osgi.contract; osgi.contract=java.servlet;
 version:Version=2.4; uses:="javax.servlet, javax.servlet.http"
```

5.1.7.2 Providing Serlvet API 2.5

```
Export-Package: javax.servlet, javax.servlet.http
Provide-Capability: osgi.contract; osgi.contract=java.servlet;
 version:Version=2.5; uses:="javax.servlet, javax.servlet.http",
 osgi.contract; osgi.contract=java.servlet;
 version:Version=2.4; uses:="javax.servlet, javax.servlet.http"
```

5.1.7.3 Providing Serlvet API 2.4

```
Export-Package: javax.servlet, javax.servlet.http
Provide-Capability: osgi.contract; osgi.contract=java.servlet;
 version:Version=2.4; uses:="javax.servlet, javax.servlet.http"
```

5.2 Whiteboard Registration Support

With Whiteboard registration support for Servlets, Listeners, Resources and 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.

Servlet, Filter, Listeners, and Resource services may register with a `osgi.http.whiteboard.service.target` property. This property is a filter expression. A Http Service about to consume a Servlet, Filter, Listener, or Resource must match that filter against his own service registration properties. Only if the filter matches, the Servlet, Filter, Listener or Resource is used by the Http Service. For example a service registered with the property

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

must only be used by an Http Service exposing the `name` property set to `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 ~~A~~ Servlet, Listener, Resource or Filter service must only be ~~registered~~used by a single Http Service. To prevent multiple ~~registration~~uses a whiteboard support implementation~~s~~ must ensure to register such objects only with a single Http Service by themselve. ~~check whether a service is already initialized by calling the `Servlet.getServletConfig()` or `Filter.getServletConfig()` method before registering the servlet or filter.~~

If more than a single whiteboard support implementation is active at runtime, there is the potential that a servlet, listener, resource or 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 matchs the filter expression.

If more than one Http Service is matching and the 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, 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.

~~Servlet, Filter, Listeners, and Resource services may register with a `osgi.http.whiteboard.service.target` property. This property is a filter expression. A Http Service about to consume a Servlet, Filter, Listener, or Resource must match that filter against his own service registration properties. Only if the filter matches, the Servlet, Filter, Listener or Resource is to be registered with the Http Service. For example a service registered with the property~~

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

~~must only be registered with an Http Service exposing the `name` property set to `admin`.~~

If more than one Http Service match, e.g. i~~I~~n the absence of the `osgi.http.whiteboard.service.target` property, any one Http Service may ~~register~~use the service. Which Http Service this is undefined.

~~Servlet, Filter, and Resource services registered with prototype scope (see RFC 195 Service Scopes) will be registered with all matching Http Services. Servlet, Filter, and Resource services registered with bundle scope will be registered with matching Http Services registered by different bundles but only with one Http Service from the same bundle.~~

The service registration properties of the Http Service ~~registering~~using the Servlet, Filter, Listener or Resource service are exposed as ServletContext initialization parameters.

5.2.1 Servlet and Filter Registration Pattern

Servlet are registered with a list of patterns in the `osgi.http.whiteboard.path` 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.

Servlet filters have been introduced into the Servlet API specification in Version 2.3 and thus far have been missing from the Http Service specification. This update adds support to register Servlets Filters the same as Servlets are registered. In addition to the patterns used for Servlet registration, a filter may be mapped to a specific Servlet by the Servlet's name.

The name of a Servlet is exposed through the `ServletConfig.getServletName()` method and is defined as follows:

- The value of the `osgi.http.whiteboard.name` service property (of type String)
- Or the fully qualified name of the Servlet implementation class

5.2.2 Http Context for Servlets, Filters and Listeners

A Servlet, Filter, Listener, or Resource can be registered with an optional `HttpContext`. In the whiteboard registration case the respective `HttpContext` must also be registered as a service and is referred to by its `osgi.http.whiteboard.context.name` service property. If the service and the Http Context referred to are registered by different bundles the Http Context must be registered with the `osgi.http.whiteboard.context.shared` property set to the boolean value `true`. Otherwise the Http Context cannot be used and the Servlet , Filter, or ~~Resource~~Listener cannot be registered.

5.2.3 Resources

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

Example:

```
Dictionary props = new Hashtable();
props.put("osgi.http.whiteboard.context.name", "resource-context");
HttpContext resourceContext = new ResourceHttpContext();
bundleContext.registerService("org.osgi.service.http.HttpContext",
                              resourceContext, props);
props = new Hashtable();
props.put("osgi.http.whiteboard.path", "/files/*");
props.put("osgi.http.whiteboard.prefix", "/tmp");
props.put("osgi.http.whiteboard.context.name", "resource-context");
bundleContext.registerService("javax.servlet.Servlet", new Resource(), props);
```

is equivalent to :

```
HttpContext resourceContext = new ResourceHttpContext();
httpService.registerResources("/files", "/tmp", resourceContext);
```

5.2.4 Event Listeners

Event listeners register themselves under the interface(s) they are implementing, supported 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 a http context like servlets and filters. The listener might specify the `osgi.http.whiteboard.context.name` service property to further define the context. The association between a listener and the context is done in the same way as described for servlets and filters.

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

5.2.4.1 ServletContextListener and ServletContextAttributeListener

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

Methods in the ServletContext object handed to the `contextInitialized` method of a registered ServletContextListener to programmatically register Servlets, 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 register Servlets, Filters, or Listeners.

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

5.2.5 Service Registration Properties

Property	Type	Description
`osgi.http.whiteboard.context.name`	String	For `HttpContext` services this property is required and identifies the service when referred to by Servlet or Filter services. Http Context services without this property are ignored. For Servlet and Filter services this refers to the Http Context services used to register the servlet or filter. If the name Http Context is missing or is registered by another bundle and does not have the `osgi.http.whiteboard.context.shared` property set to `true`, the servlet or filter cannot be registered.
`osgi.http.whiteboard.context.shared`	Boolean	Whether a Http Context service may be used by Servlet or Filter services registered by other bundles. By default Http Context services can only be used by

		Servlet and Filter services registered by the same bundle.
`osgi.http.whiteboard.name`	String	The name of a Servlet or Filter registered. This name is used as the value of the `ServletConfig.getServletName()` and `FilterConfig.getFilterName()` methods and defaults to the fully qualified name of the service object's class. Filter services may refer to servlets by this name in their `osgi.http.whiteboard.path` property to apply the filter to a concrete servlet.
`osgi.http.whiteboard.path`	String+	Registration pattern for the Servlet or Filter. See section 5.2.1, Servlet and Filter Registration Pattern for a description. Servlet or Filter services not registered with this property are ignored.
`osgi.http.whiteboard.asyncSupport`	Boolean	Declares whether the servlet or filter supports asynchronous operation mode.
`osgi.http.whiteboard.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. This property is only used for Servlet services actually registered. For this property to be applicable the servlet must be registered with a single value `osgi.http.whiteboard.pattern` property being an absolute path. Otherwise the `osgi.http.whiteboard.errorPage` property must be ignored.
`osgi.http.whiteboard.service.target`	String	Servlet or Filter services registered with this property are registered with an Http Service whose service registration properties match this LDAP filter expression.
`osgi.http.whiteboard.prefix`	String	Registers a mapping from the prefix pattern defined in the single valued `osgi.http.whiteboard.path` to resources found at the given prefix.

Note: *String+* means `String`, `String[]` or `Collection<String>`.

5.2.6 Provided Capability

The bundle implementing whiteboard Servlet, Filter, Listener and HttpContext support has to provide the following `osgi.whiteboard` capability:

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

5.2.6.1 osgi.whiteboard Namespace

The whiteboard pattern leverages the OSGi service registry as a registry for objects. In the context of Http Service servlets could be registered directly with the Http Service and the Http Service could manage the registry. Applying the whiteboard pattern the services are registered with the OSGi service registry.

A Whiteboard Consumer is a bundle that uses the life cycle events from another bundle, the *extendee,* to extend that bundle's functionality when that bundle is active. It can use metadata (headers, or files inside the extendee)

to control its functionality. Whiteboard Providers therefore have a dependency on the Whiteboard 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 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 consumer.

Specifications for whiteboard consumers (Http Service, Event Admin, etc.) should specify the values for these attributes. Whiteboard consumers that provide such a capability should list the packages that they use in their specification in the uses directive of that capability to ensure class space consistency. Whiteboard consumers can consume a whiteboard 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 a Http Service could require such a whiteboard consumer with the following manifest header:

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

6 Data Transfer Objects

This chapter defines an API to retrieve administrative information from the Http Service. The DTOs are accessed through the Http Service interface:

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

See the JavaDoc for details.

7 Javadoc

OSGi Javadoc

08.08.13 17:18

Package Summary		Page
org.osgi.service.http	Http Service Package Version 1.3.	21
org.osgi.service.http.dto	OSGi Data Transfer Object Http Service Package Version 1.3.	35

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	This interface defines methods that the Http Service may call to get information about a registration.	*25*
HttpService	The Http Service allows other bundles in the OSGi environment to dynamically register resources and servlets into the URI namespace of Http Service.	*28*

Class Summary		Page
HttpConstants	Defines standard names for Http Service constants.	*22*
Resource	The `Resource` servlet is a marker servlet which can be used to register resources through the Whiteboard to serve resources through the Http Context with which the Resource is registered.	*34*

Exception Summary		Page
NamespaceException	**Deprecated.** *as of 1.3*	*32*

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
Version:
$Id: 4b7be7d793964cb6471df2af7d2cf2da4985e20b $

Field Summary		Page
static String	**HTTP_SERVICE_ENDPOINTS** Lists URLs to which the Http Service is bound; e.g.	24
static String	**HTTP_WHITEBOARD_CONTEXT_NAME** For HttpContext services this property is required and identifies the service when referred to by Servlet or Filter services.	22
static String	**HTTP_WHITEBOARD_CONTEXT_SHARED** Whether a Http Context service may be used by Servlet or Filter services registered by other bundles.	23
static String	**HTTP_WHITEBOARD_ERROR_PAGE** 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.	23
static String	**HTTP_WHITEBOARD_NAME** The name of a Servlet or Filter registered.	23
static String	**HTTP_WHITEBOARD_PATH** Registration pattern for the Servlet or Filter.	23
static String	**HTTP_WHITEBOARD_PREFIX** Registers a mapping from the prefix pattern defined in the single valued HTTP_WHITEBOARD_PATH to resources found at the given prefix.	24
static String	**HTTP_WHITEBOARD_TARGET** Servlet or Filter services registered with this property are registered with an Http Service whose service registration properties match this LDAP filter expression.	23

Field Detail

HTTP_WHITEBOARD_CONTEXT_NAME

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

For HttpContext services this property is required and identifies the service when referred to by Servlet or Filter services. Http Context services without this property are ignored. For Servlet and Filter services this refers to the Http Context services used to register the servlet or filter. If the name Http Context is missing or is registered by another bundle and does not have the HTTP_WHITEBOARD_CONTEXT_SHARED property set to true, the servlet or filter cannot be registered.

The type of this servlet registration property is String.

See Also:
HTTP_WHITEBOARD_CONTEXT_SHARED

HTTP_WHITEBOARD_CONTEXT_SHARED

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

Whether a Http Context service may be used by Servlet or Filter services registered by other bundles. By default Http Context services can only be used by Servlet and Filter services registered by the same bundle.

The type of this servlet registration property is Boolean.

See Also:
HTTP_WHITEBOARD_CONTEXT_NAME

HTTP_WHITEBOARD_NAME

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

The name of a Servlet or Filter registered. This name is used as the value of the `ServletConfig.getServletName()` and `FilterConfig.getFilterName()` methods and defaults to the fully qualified name of the service object's class. Filter services may refer to servlets by this name in their HTTP_WHITEBOARD_PATH property to apply the filter to a concrete servlet.

The type of this servlet registration property is String.

See Also:
HTTP_WHITEBOARD_PATH

HTTP_WHITEBOARD_PATH

```
public static final String HTTP_WHITEBOARD_PATH = "osgi.http.whiteboard.path"
```

Registration pattern for the Servlet or Filter. See section 5.1.2, Servlet and Filter Registration for a description. Servlet or Filter services not registered with this property are ignored.

The type of this servlet registration property is String+.

HTTP_WHITEBOARD_ERROR_PAGE

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

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. This property is only used for Servlet services actually registered. For this property to be applicable the servlet must be registered with a single value HTTP_WHITEBOARD_PATH property being an absolute path. Otherwise this property must be ignored.

The type of this servlet registration property is String+.

See Also:
HTTP_WHITEBOARD_PATH

HTTP_WHITEBOARD_TARGET

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

Servlet or Filter services registered with this property are registered with an Http Service whose service registration properties match this LDAP filter expression.

The type of this servlet registration property is String.

HTTP_WHITEBOARD_PREFIX

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

Registers a mapping from the prefix pattern defined in the single valued HTTP_WHITEBOARD_PATH to resources found at the given prefix.

The type of this servlet registration property is String.

See Also:
HTTP_WHITEBOARD_PATH

HTTP_SERVICE_ENDPOINTS

```
public static final String HTTP_SERVICE_ENDPOINTS = "osgi.http.service.endpoints"
```

Lists URLs to which the Http Service is bound; e.g. http://192.168.1.10:8080/. The relevant information contained in the URLs is the scheme, IP Address of the bound interface, bound port, and the (optional) context path in a Servlet API servlet container the Http Service is registered.

The type of this servlet registration property is String+.

Interface HttpContext

org.osgi.service.http

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

This interface defines methods that the Http Service may call to get information about a registration.

Servlets and resources may be registered with an `HttpContext` object; if no `HttpContext` object is specified, a default `HttpContext` object is used. Servlets that are registered using the same `HttpContext` object will share the same `ServletContext` object.

This interface is implemented by users of the `HttpService`.

Field Summary		Page
String	**AUTHENTICATION_TYPE** `HttpServletRequest` attribute specifying the scheme used in authentication.	25
String	**AUTHORIZATION** `HttpServletRequest` attribute specifying the `Authorization` object obtained from the `org.osgi.service.useradmin.UserAdmin` service.	26
String	**REMOTE_USER** `HttpServletRequest` attribute specifying the name of the authenticated user.	25

Method Summary		Page
String	**getMimeType**(String name) Maps a name to a MIME type.	27
URL	**getResource**(String name) Maps a resource name to a URL.	27
boolean	**handleSecurity**(HttpServletRequest request, HttpServletResponse response) Handles security for the specified request.	26

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 a resource name to a URL. For servlet registrations, Http Service will call this method to support the `ServletContext` methods `getResource` and `getResourceAsStream`. For resource registrations, 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:
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 name. For servlet registrations, the Http Service will call this method to support the `ServletContext` method `getMimeType`. For resource registrations, the Http Service will call this method to determine the MIME type for the Content-Type header in the response.

Parameters:
`name` - determine the MIME type for this name.
Returns:
MIME type (e.g. text/html) of the 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
```

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

Method Summary		Page
HttpContext	**createDefaultHttpContext**() **Deprecated.** *as of 1.3 with no replacement*	30
Map<String,String>	**getErrorLocations**()	31
FilterDTO[]	**getFilters**()	31
Map<String,String>	**getResources**() Returns the registered resources as a map indexed by the resource alias with the resource prefixes being the values.	31
ServletDTO[]	**getServlets**()	30
void	**registerResources**(String alias, String name, HttpContext context) **Deprecated.** *as of 1.3, use the whiteboard pattern registration of resources: Register a Resource instance as a service of type javax.servlet.Servlet adding at least the HttpConstants.HTTP_WHITEBOARD_PATH and HttpConstants.HTTP_WHITEBOARD_PREFIX service registration properties.*	29
void	**registerServlet**(String alias, Servlet servlet, Dictionary<String,String> initparams, HttpContext context) **Deprecated.** *as of 1.3, use the whiteboard pattern registration of servlets: Register the servlet as a service of type javax.servlet.Servlet adding at least the HttpConstants.HTTP_WHITEBOARD_PATH service registration property.*	28
void	**unregister**(String alias) **Deprecated.** *as of 1.3 this method can only unregister servlets and resource registered through the deprecated registerServlet(String, Servlet, Dictionary, HttpContext) and registerResources(String, String, HttpContext) methods.*	30

Method Detail

registerServlet

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

Deprecated. *as of 1.3, use the whiteboard pattern registration of servlets: Register the servlet as a service of type javax.servlet.Servlet adding at least the HttpConstants.HTTP_WHITEBOARD_PATH service registration property.*

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
```

Deprecated. *as of 1.3, use the whiteboard pattern registration of resources: Register a `Resource` instance as a service of type `javax.servlet.Servlet` adding at least the `HttpConstants.HTTP_WHITEBOARD_PATH` and `HttpConstants.HTTP_WHITEBOARD_PREFIX` service registration properties.*

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

```
@Deprecated
void unregister(String alias)
```

Deprecated. *as of 1.3 this method can only unregister servlets and resource registered through the deprecated registerServlet(String, Servlet, Dictionary, HttpContext) and registerResources(String, String, HttpContext) methods.*

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()
```

Deprecated. *as of 1.3 with no replacement*

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:

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 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

---

getServlets

```
ServletDTO[] getServlets()
```

Returns:
the registered servlets or `null` if no servlets are registered.
Since:
1.3
See Also:
registerServlet(String, Servlet, Dictionary, HttpContext)

getResources

`Map<String,String>` **`getResources`**`()`

Returns the registered resources as a map indexed by the resource alias with the resource prefixes being the values.

Returns:
the registered resources or `null` if no resources are registered.
Since:
1.3
See Also:
`registerResources(String, String, HttpContext)`

getFilters

`FilterDTO[]` **`getFilters`**`()`

Returns:
the registered filters or `null` if no filters are registered.
Since:
1.3

getErrorLocations

`Map<String,String>` **`getErrorLocations`**`()`

Returns:
defined error location mappings or `null` if no error location mappings have been registered.
Since:
1.3

Class NamespaceException

org.osgi.service.http

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

All Implemented Interfaces:
Serializable

```
@Deprecated
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.	32
`NamespaceException(String message, Throwable cause)` Construct a `NamespaceException` object with a detail message and a nested exception.	32

Method Summary		Page
`Throwable`	`getCause()` Returns the cause of this exception or `null` if no cause was set.	33
`Throwable`	`getException()` Returns the nested exception.	33
`Throwable`	`initCause(Throwable cause)` Initializes the cause of this exception to the specified value.	33

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 Resource

org.osgi.service.http

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

All Implemented Interfaces:
Serializable, Servlet, ServletConfig

```
final public class Resource
extends GenericServlet
```

The `Resource` servlet is a marker servlet which can be used to register resources through the Whiteboard to serve resources through the Http Context with which the Resource is registered.

The `Resource` object is registered as a `javax.servlet.Servlet` service along with the `HttpConstants.HTTP_WHITEBOARD_PATH`, `HttpConstants.HTTP_WHITEBOARD_PREFIX`, and `HttpConstants.HTTP_WHITEBOARD_CONTEXT_NAME` properties. See the specification text for more details.

`Resource` extends from `GenericServlet` and implements the `service` method with an empty method. Therefore calling the `service` method on a resource has absolutely no effect.

The default constructor must be used to instantiate new objects, the constructor as well is empty and does not call any other method.

Since:
1.3
Version:
$Id: 0b578492d62dc4e4f1d5a96399d2de4c43ec9e74 $

Constructor Summary	Page
Resource()	34

Method Summary		Page
void	**service**(ServletRequest request, ServletResponse response)	34

Constructor Detail

Resource

```
public Resource()
```

Method Detail

service

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

Specified by:
`service` in interface `Servlet`
Overrides:
`service` in class `GenericServlet`

Package org.osgi.service.http.dto

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

OSGi Data Transfer Object Http Service Package Version 1.3.

See:
Description

Class Summary		Page
FilterDTO	Represents a `Filter` registered with the `HttpService`.	36
ServletContextDTO	Represents a `ServletContext` created for registered servlets and filters backed by the `HttpContext` objects used during registration.	38
ServletDTO	Represents a `Servlet` registered with the `HttpService`.	40

Package org.osgi.service.http.dto Description

OSGi Data Transfer Object 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.dto; version="[1.3,2.0)"
```

Example import for providers implementing the API in this package:

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

Class FilterDTO

org.osgi.service.http.dto

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

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

Represents a `Filter` registered with the `HttpService`.

Version:
$Id: c9e5922919c8ff6038850dc5ed2711a0423e1b3e $

Field Summary		Page
String	**name** The name of the filter as returned by the `FilterConfig.getFilterName()` method.	36
String[]	**paths** The value of the `HttpConstants.HTTP_WHITEBOARD_PATH` service registration property of the registered `javax.servlet.Filter` service.	36
ServletContextDTO	**servletContext** The `ServletDTO` representing the `ServletContext` to which this filter is registered.	37

Constructor Summary	Page
FilterDTO()	37

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

Field Detail

paths

```
public String[] paths
```

The value of the `HttpConstants.HTTP_WHITEBOARD_PATH` service registration property of the registered `javax.servlet.Filter` service.

See Also:
`HttpConstants.HTTP_WHITEBOARD_PATH`

---

name

```
public String name
```

The name of the filter as returned by the `FilterConfig.getFilterName()` method.

See Also:
`HttpConstants.HTTP_WHITEBOARD_NAME`

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletDTO representing the ServletContext to which this filter is registered.

Constructor Detail

FilterDTO

```
public FilterDTO()
```

Class ServletContextDTO

org.osgi.service.http.dto

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

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

Represents a `ServletContext` created for registered servlets and filters backed by the `HttpContext` objects used during registration.

Version:
$Id: 1a4d0af68f98f053eb5ec5bd14ec3e58d4043027 $

Field Summary		Page
`Map<String,Object>`	**attributes** Servlet context attributes The value type must be a numerical type, Boolean, String, DTO or an array of any of the former.	39
`String`	**contextPath** The servlet context path returned from the `ServletContext.getContextPath()` method.	38
`Map<String,String>`	**initParams** Servlet context initialization parameters	39
`String`	**name** The name of the servlet context returned from the `ServletContext.getServletContextName()`.	38

Constructor Summary	Page
ServletContextDTO()	39

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

Field Detail

name

```
public String name
```

The name of the servlet context returned from the `ServletContext.getServletContextName()`.

---

contextPath

```
public String contextPath
```

The servlet context path returned from the `ServletContext.getContextPath()` method.

---

initParams

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

Servlet context initialization parameters

---

attributes

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

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.dto

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

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

Represents a `Servlet` registered with the `HttpService`.

Version:
$Id: 25c610ec71badfe96c9db47f4760322db6c61d3a $

Field Summary		Page
String	**name** The name of the servlet as returned by the `ServletConfig.getServletName()` method.	41
String[]	**paths** The URL alias used to register the servlet with the `HttpService.registerServlet(String, javax.servlet.Servlet, java.util.Dictionary, org.osgi.service.http.HttpContext)` method or the value of the `HttpConstants.HTTP_WHITEBOARD_PATH` service registration property of the registered `javax.servlet.Servlet` service.	40
ServletContextDTO	**servletContext** The `ServletDTO` representing the `ServletContext` to which this servlet is registered.	41
String	**servletInfo** The servlet information string returned from the servlet through the `Servlet.getServletInfo()` method.	41

Constructor Summary	Page
ServletDTO()	41

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

Field Detail

paths

```
public String[] paths
```

The URL alias used to register the servlet with the `HttpService.registerServlet(String, javax.servlet.Servlet, java.util.Dictionary, org.osgi.service.http.HttpContext)` method or the value of the `HttpConstants.HTTP_WHITEBOARD_PATH` service registration property of the registered `javax.servlet.Servlet` service.

See Also:
`HttpService.registerServlet(String, javax.servlet.Servlet, java.util.Dictionary, org.osgi.service.http.HttpContext)`, `HttpConstants.HTTP_WHITEBOARD_PATH`

---

name

```
public String name
```

The name of the servlet as returned by the `ServletConfig.getServletName()` method.

See Also:
HttpConstants.HTTP_WHITEBOARD_NAME

---

servletInfo

```
public String servletInfo
```

The servlet information string returned from the servlet through the `Servlet.getServletInfo()` method.

---

servletContext

```
public ServletContextDTO servletContext
```

The ServletDTO representing the `ServletContext` to which this servlet is registered.

Constructor Detail

ServletDTO

```
public ServletDTO()
```

OSGi Javadoc

28.01.13 21:03

Package Summary		Page
org.osgi.namespace.whiteboard	Whiteboard Namespace Package Version 1.0.	21

Package org.osgi.namespace.whiteboard

Whiteboard Namespace Package Version 1.0.

See:
Description

Class Summary		Page
WhiteboardNamespace	Whiteboard Capability and Requirement Namespace.	23

Package org.osgi.namespace.whiteboard Description

Whiteboard Namespace Package Version 1.0.

Bundles should not need to import this package at runtime since all the types in this package just contain constants for capability and requirement namespaces specified by the OSGi Alliance.

Class WhiteboardNamespace

org.osgi.namespace.whiteboard

```
java.lang.Object
  └ org.osgi.resource.Namespace
      └ org.osgi.namespace.whiteboard.WhiteboardNamespace
```

```
final public class WhiteboardNamespace
extends org.osgi.resource.Namespace
```

Whiteboard Capability and Requirement Namespace.

This class defines the names for the attributes and directives for this namespace. All unspecified capability attributes are of type `String` and are used as arbitrary matching attributes for the capability. The values associated with the specified directive and attribute keys are of type `String`, unless otherwise indicated.

Version:
$Id: b23440f15fa24e7c0b0e27896d234d2fc4ecbb3a $
Immutable

Field Summary		Page
static String	**CAPABILITY_BUNDLE_SYMBOLICNAME_ATTRIBUTE** The capability attribute contains the symbolic name of the bundle providing the whiteboard consumer.	23
static String	**CAPABILITY_BUNDLE_VERSION_ATTRIBUTE** The capability attribute contains the `Version` of the bundle implementing the whiteboard consumer if one is specified or `0.0.0` if not specified.	23
static String	**CAPABILITY_VERSION_ATTRIBUTE** The capability attribute contains the `Version` of the specification of the whiteboard consumer.	23
static String	**WHITEBOARD_NAMESPACE** Namespace name for whiteboard capabilities and requirements.	22

Fields inherited from class org.osgi.resource.Namespace
CAPABILITY_EFFECTIVE_DIRECTIVE, CAPABILITY_USES_DIRECTIVE, CARDINALITY_MULTIPLE, CARDINALITY_SINGLE, EFFECTIVE_ACTIVE, EFFECTIVE_RESOLVE, REQUIREMENT_CARDINALITY_DIRECTIVE, REQUIREMENT_EFFECTIVE_DIRECTIVE, REQUIREMENT_FILTER_DIRECTIVE, REQUIREMENT_RESOLUTION_DIRECTIVE, RESOLUTION_MANDATORY, RESOLUTION_OPTIONAL

Field Detail

WHITEBOARD_NAMESPACE

```
public static final String WHITEBOARD_NAMESPACE = "osgi.whiteboard"
```

Namespace name for whiteboard capabilities and requirements.

Also, the capability attribute used to specify the name of the whiteboard consumer.

CAPABILITY_VERSION_ATTRIBUTE

```
public static final String CAPABILITY_VERSION_ATTRIBUTE = "version"
```

The capability attribute contains the `Version` of the specification of the whiteboard consumer. The value of this attribute must be of type `Version`.

CAPABILITY_BUNDLE_VERSION_ATTRIBUTE

```
public static final String CAPABILITY_BUNDLE_VERSION_ATTRIBUTE = "bundle-version"
```

The capability attribute contains the `Version` of the bundle implementing the whiteboard consumer if one is specified or `0.0.0` if not specified. The value of this attribute must be of type `Version`.

CAPABILITY_BUNDLE_SYMBOLICNAME_ATTRIBUTE

```
public static final String CAPABILITY_BUNDLE_SYMBOLICNAME_ATTRIBUTE = "bundle-symbolic-name"
```

The capability attribute contains the symbolic name of the bundle providing the whiteboard consumer.

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.

9 Security Considerations

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