102 Http Service Specification

Version 1.2

102.1 Introduction

An OSGi Service Platform normally provides users with access to services on the Internet and other networks. This access allows users to remotely retrieve information from, and send control to, services in an OSGi Service Platform using a standard web browser.

Bundle developers typically need to develop communication and user interface solutions for standard technologies such as HTTP, HTML, XML, and servlets.

The Http Service supports two standard techniques for this purpose:

- *Registering servlets* – A servlet is a Java object which implements the Java Servlet API. Registering a servlet in the Framework gives it control over some part of the Http Service URI name-space.
- *Registering resources* – Registering a resource allows HTML files, image files, and other static resources to be made visible in the Http Service URI name-space by the requesting bundle.

Implementations of the Http Service can be based on:

- [1] *HTTP 1.0 Specification RFC-1945*
- [2] *HTTP 1.1 Specification RFC-2616*

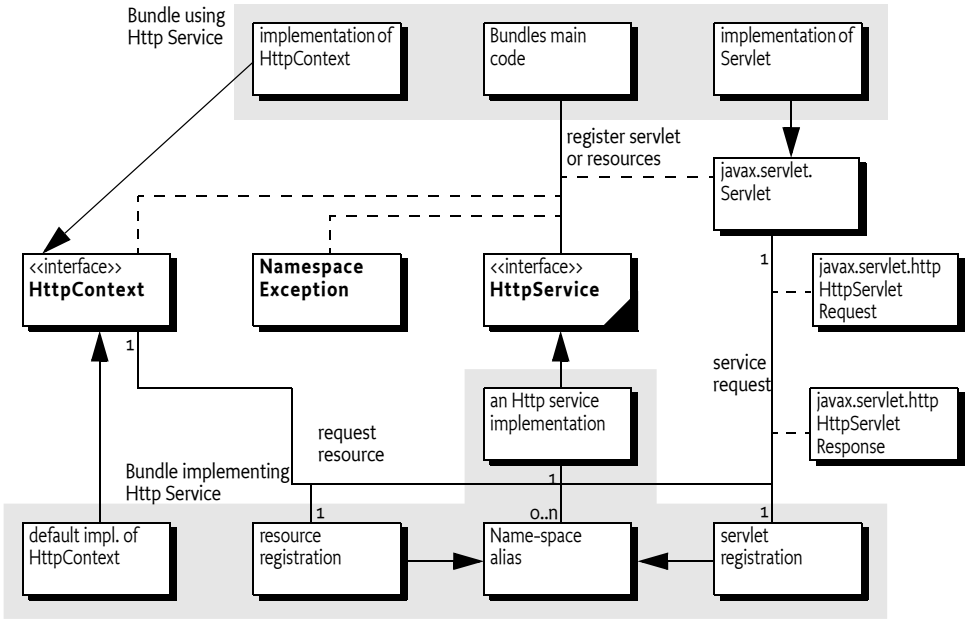
Alternatively, implementations of this service can support other protocols if these protocols can conform to the semantics of the javax.servlet API. This additional support is necessary because the Http Service is closely related to [3] *Java Servlet Technology*. Http Service implementations must support at least version 2.1 of the Java Servlet API.

102.1.1 Entities

This specification defines the following interfaces which a bundle developer can implement collectively as an Http Service or use individually:

- HttpContext – Allows bundles to provide information for a servlet or resource registration.
- HttpService – Allows other bundles in the Framework to dynamically register and unregister resources and servlets into the Http Service URI name-space.
- NamespaceException – Is thrown to indicate an error with the caller's request to register a servlet or resource into the Http Service URI name-space.

Figure 102.1 Http Service Overview Diagram



102.2 Registering Servlets

javax.servlet.Servlet objects can be registered with the Http Service by using the HttpService interface. For this purpose, the HttpService interface defines the method registerServlet(String, javax.servlet.Servlet,Dictionary,HttpContext).

For example, if the Http Service implementation is listening to port 80 on the machine www.acme.com and the Servlet object is registered with the name "/servlet", then the Servlet object's service method is called when the following URL is used from a web browser:

```
http://www.acme.com/servletname=bugs
```

All Servlet objects and resource registrations share the same name-space. If an attempt is made to register a resource or Servlet object under the same name as a currently registered resource or Servlet object, a NamespaceException is thrown. See *Mapping HTTP Requests to Servlet and Resource Registrations* on page 33 for more information about the handling of the Http Service name-space.

Each Servlet registration must be accompanied with an HttpContext object. This object provides the handling of resources, media typing, and a method to handle authentication of remote requests. See *Authentication* on page 36.

For convenience, a default HttpContext object is provided by the Http Service and can be obtained with createDefaultHttpContext(). Passing a null parameter to the registration method achieves the same effect.

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.

Servlet objects are initialized by the Http Service when they are registered and bound to that specific Http Service. The initialization is done by calling the Servlet object's Servlet.init(ServletConfig) method. The ServletConfig parameter provides access to the initialization parameters specified when the Servlet object was registered.

Therefore, the same Servlet instance must not be reused for registration with another Http Service, nor can it be registered under multiple names. Unique instances are required for each registration.

The following example code demonstrates the use of the registerServlet method:

```
Hashtable initparams = new Hashtable();
initparams.put( "name", "value" );

Servlet myServlet = new HttpServlet() {
   String     name = "<not set>";

   public void init( ServletConfig config ) {
      this.name = (String)
         config.getInitParameter( "name" );
   }

   public void doGet(
      HttpServletRequest req,
      HttpServletResponse rsp
   ) throws IOException {
      rsp.setContentType( "text/plain" );
      req.getWriter().println( this.name );
   }
};

getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   null // use default context
);
// myServlet has been registered
// and its init method has been called. Remote
// requests are now handled and forwarded to
// the servlet.
...
getHttpService().unregister("/servletAlias");
// myServlet has been unregistered and its
// destroy method has been called
```

This example registers the servlet, myServlet, at alias: /servletAlias. Future requests for http://www.acme.com/servletAlias maps to the servlet, myServlet, whose service method is called to process the request. (The service method is called in the HttpServlet base class and dispatched to a doGet, doPut, doPost, doOptions, doTrace, or doDelete call depending on the HTTP request method used.)

102.3 Registering Resources

A resource is a file containing images, static HTML pages, sounds, movies, applets, etc. Resources do not require any handling from the bundle. They are transferred directly from their source–usually the JAR file that contains the code for the bundle–to the requestor using HTTP.

Resources could be handled by Servlet objects as explained in *Registering Servlets* on page 30. Transferring a resource over HTTP, however, would require very similar Servlet objects for each bundle. To prevent this redundancy, resources can be registered directly with the Http Service via the HttpService interface. This HttpService interface defines the registerResources(String,String,HttpContext)method for registering a resource into the Http Service URI name-space.

The first parameter is the external alias under which the resource is registered with the Http Service. The second parameter is an internal prefix to map this resource to the bundle's name-space. When a request is received, the HttpService object must remove the external alias from the URI, replace it with the internal prefix, and call the getResource(String) method with this new name on the associated HttpContext object. The HttpContext object is further used to get the MIME type of the resource and to authenticate the request.

Resources are returned as a java.net.URL object. The Http Service must read from this URL object and transfer the content to the initiator of the HTTP request.

This return type was chosen because it matches the return type of the java.lang.Class.getResource(String resource) method. This method can retrieve resources directly from the same place as the one from which the class was loaded – often a package directory in the JAR file of the bundle. This method makes it very convenient to retrieve resources from the bundle that are contained in the package.

The following example code demonstrates the use of the registerResources method:

```
package com.acme;
...
HttpContext context = new HttpContext() {
   public boolean handleSecurity(
      HttpServletRequest request,
      HttpServletResponse response
   ) throws IOException {
      return true;
   }

   public URL getResource(String name) {
      return getClass().getResource(name);
   }

   public String getMimeType(String name) {
      return null;
   }
};

getHttpService().registerResources (
   "/files",
   "www",
   context
);
...
getHttpService().unregister("/files");
```

This example registers the alias /files on the Http Service. Requests for resources below this name-space are transferred to the HttpContext object with an internal name of www/<name>. This example uses the Class.getResource(String) method. Because the internal name does not start with a

"/", it must map to a resource in the "com/acme/www" directory of the JAR file. If the internal name did start with a "/", the package name would not have to be prefixed and the JAR file would be searched from the root. Consult the java.lang.Class.getResource(String) method for more information.

In the example, a request for http://www.acme.com/files/myfile.html must map to the name "com/acme/www/myfile.html" which is in the bundle's JAR file.

More sophisticated implementations of the getResource(String) method could filter the input name, restricting the resources that may be returned or map the input name onto the file system (if the security implications of this action are acceptable).

Alternatively, the resource registration could have used a default HttpContext object, as demonstrated in the following call to registerResources:

```
getHttpService().registerResources(
   "/files",
   "/com/acme/www",
   null
);
```

In this case, the Http Service implementation would call the createDefaultHttpContext() method and use its return value as the HttpContext argument for the registerResources method. The default implementation must map the resource request to the bundle's resource, using Bundle.getResource(String). In the case of the previous example, however, the internal name must now specify the full path to the directory containing the resource files in the JAR file. No automatic prefixing of the package name is done.

The getMimeType(String) implementation of the default HttpContext object should rely on the default mapping provided by the Http Service by returning null. Its handleSecurity(HttpServletRequest,HttpServletResponse) may implement an authentication mechanism that is implementation-dependent.

102.4 Mapping HTTP Requests to Servlet and Resource Registrations

When an HTTP request comes in from a client, the Http Service checks to see if the requested URI matches any registered aliases. A URI matches only if the path part of the URI is exactly the same string. Matching is case sensitive.

If it does match, a matching registration takes place, which is processed as follows:

1. If the registration corresponds to a servlet, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See *Authentication* on page 36. If the request is authorized, the servlet must be called by its service method to complete the HTTP request.
2. If the registration corresponds to a resource, the authorization is verified by calling the handleSecurity method of the associated HttpContext object. See *Authentication* on page 36. If the request is authorized, a target resource name is constructed from the requested URI by substituting the alias from the registration with the internal name from the registration if the alias is not "/". If the alias is "/", then the target resource name is constructed by prefixing the requested URI with the internal name. An internal name of "/" is considered to have the value of the empty string ("") during this process.
3. The target resource name must be passed to the getResource method of the associated HttpContext object.

4. If the returned URL object is not null, the Http Service must return the contents of the URL to the client completing the HTTP request. The translated target name, as opposed to the original requested URI, must also be used as the argument to HttpContext.getMimeType.
5. If the returned URL object is null, the Http Service continues as if there was no match.
6. If there is no match, the Http Service must attempt to match sub-strings of the requested URI to registered aliases. The sub-strings of the requested URI are selected by removing the last "/" and everything to the right of the last "/".

The Http Service must repeat this process until either a match is found or the sub-string is an empty string. If the sub-string is empty and the alias "/" is registered, the request is considered to match the alias "/". Otherwise, the Http Service must return HttpServletResponse.SC_NOT_FOUND(404) to the client.

For example, an HTTP request comes in with a request URI of "/fudd/bugs/foo.txt", and the only registered alias is "/fudd". A search for "/fudd/bugs/foo.txt" will not match an alias. Therefore, the Http Service will search for the alias "/fudd/bugs" and the alias "/fudd". The latter search will result in a match and the matched alias registration must be used.

Registrations for identical aliases are not allowed. If a bundle registers the alias "/fudd", and another bundle tries to register the exactly the same alias, the second caller must receive a NamespaceException and its resource or servlet must *not* be registered. It could, however, register a similar alias – for example, "/fudd/bugs", as long as no other registration for this alias already exists.

The following table shows some examples of the usage of the name-space.

Table 102.1 Examples of Name-space Mapping

Alias	Internal Name	URI	getResource Parameter
/	(empty string)	/fudd/bugs	/fudd/bugs
/	/	/fudd/bugs	/fudd/bugs
/	/tmp	/fudd/bugs	/tmp/fudd/bugs
/fudd	(empty string)	/fudd/bugs	/bugs
/fudd	/	/fudd/bugs	/bugs
/fudd	/tmp	/fudd/bugs	/tmp/bugs
/fudd	tmp	/fudd/bugs/x.gif	tmp/bugs/x.gif
/fudd/bugs/x.gif	tmp/y.gif	/fudd/bugs/x.gif	tmp/y.gif

102.5 The Default Http Context Object

The HttpContext object in the first example demonstrates simple implementations of the HttpContext interface methods. Alternatively, the example could have used a default HttpContext object, as demonstrated in the following call to registerServlet:

```
getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   null
);
```

In this case, the Http Service implementation must call createDefault HttpContext and use the return value as the HttpContext argument.

If the default HttpContext object, and thus the ServletContext object, is to be shared by multiple servlet registrations, the previous servlet registration example code needs to be changed to use the same default HttpContext object. This change is demonstrated in the next example:

```
HttpContext defaultContext =
   getHttpService().createDefaultHttpContext();

getHttpService().registerServlet(
   "/servletAlias",
   myServlet,
   initparams,
   defaultContext
);

// defaultContext can be reused
// for further servlet registrations
```

102.6 Multipurpose Internet Mail Extension (MIME) Types

MIME defines an extensive set of headers and procedures to encode binary messages in US-ASCII mails. For an overview of all the related RFCs, consult [4] *MIME Multipurpose Internet Mail Extension.*

An important aspect of this extension is the type (file format) mechanism of the binary messages. The type is defined by a string containing a general category (text, application, image, audio and video, multipart, and message) followed by a "/" and a specific media type, as in the example, "text/html" for HTML formatted text files. A MIME type string can be followed by additional specifiers by separating key=value pairs with a ';'. These specifiers can be used, for example, to define character sets as follows:

```
text/plain ; charset=iso-8859-1
```

The Internet Assigned Number Authority (IANA) maintains a set of defined MIME media types. This list can be found at [5] *Assigned MIME Media Types.* MIME media types are extendable, and when any part of the type starts with the prefix "x-", it is assumed to be vendor-specific and can be used for testing. New types can be registered as described in [6] *Registration Procedures for new MIME media types.*

HTTP bases its media typing on the MIME RFCs. The "Content-Type" header should contain a MIME media type so that the browser can recognize the type and format the content correctly.

The source of the data must define the MIME media type for each transfer. Most operating systems do not support types for files, but use conventions based on file names, such as the last part of the file name after the last ".". This extension is then mapped to a media type.

Implementations of the Http Service should have a reasonable default of mapping common extensions to media types based on file extensions.

Table 102.2 Sample Extension to MIME Media Mapping

Extension	MIME media type	Description
.jpg .jpeg	image/jpeg	JPEG Files
.gif	image/gif	GIF Files
.css	text/css	Cascading Style Sheet Files
.txt	text/plain	Text Files
.wml	text/vnd.wap.wml	Wireless Access Protocol (WAP) Mark Language

Table 102.2 Sample Extension to MIME Media Mapping

Extension	MIME media type	Description
.htm .html	text/html	Hyper Text Markup Language
.wbmp	image/vnd.wap.wbmp	Bitmaps for WAP

Only the bundle developer, however, knows exactly which files have what media type. The HttpContext interface can therefore be used to map this knowledge to the media type. The HttpContext class has the following method for this: getMimeType(String).

The implementation of this method should inspect the file name and use its internal knowledge to map this name to a MIME media type.

Simple implementations can extract the extension and look up this extension in a table.

Returning null from this method allows the Http Service implementation to use its default mapping mechanism.

102.7 Authentication

The Http Service has separated the authentication and authorization of a request from the execution of the request. This separation allows bundles to use available Servlet sub-classes while still providing bundle specific authentication and authorization of the requests.

Prior to servicing each incoming request, the Http Service calls the handleSecurity(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse) method on the HttpContext object that is associated with the request URI. This method controls whether the request is processed in the normal manner or an authentication error is returned.

If an implementation wants to authenticate the request, it can use the authentication mechanisms of HTTP. See [7] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. These mechanisms normally interpret the headers and decide if the user identity is available, and if it is, whether that user has authenticated itself correctly.

There are many different ways of authenticating users, and the handleSecurity method on the HttpContext object can use whatever method it requires. If the method returns true, the request must continue to be processed using the potentially modified HttpServletRequest and HttpServletResponse objects. If the method returns false, the request must *not* be processed.

A common standard for HTTP is the basic authentication scheme that is not secure when used with HTTP. Basic authentication passes the password in base 64 encoded strings that are trivial to decode into clear text. Secure transport protocols like HTTPS use SSL to hide this information. With these protocols basic authentication is secure.

Using basic authentication requires the following steps:

1. If no Authorization header is set in the request, the method should set the WWW-Authenticate header in the response. This header indicates the desired authentication mechanism and the realm. For example, WWW-Authenticate: Basic realm="ACME".
   The header should be set with the response object that is given as a parameter to the handleSecurity method. The handleSecurity method should set the status to HttpServletResponse.SC_UNAUTHORIZED (401) and return false.
2. Secure connections can be verified with the ServletRequest.getScheme() method. This method returns, for example, "https" for an SSL connection; the handleSecurity method can use this and other information to decide if the connection's security level is acceptable. If not, the handleSecurity method should set the status to HttpServletResponse.SC_FORBIDDEN (403) and return false.

3. Next, the request must be authenticated. When basic authentication is used, the Authorization header is available in the request and should be parsed to find the user and password. See [7] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication* for more information.
   If the user cannot be authenticated, the status of the response object should be set to HttpServletResponse.SC_UNAUTHORIZED (401) and return false.
4. The authentication mechanism that is actually used and the identity of the authenticated user can be of interest to the Servlet object. Therefore, the implementation of the handleSecurity method should set this information in the request object using the ServletRequest.setAttribute method. This specification has defined a number of OSGi-specific attribute names for this purpose:
   - AUTHENTICATION_TYPE - Specifies the scheme used in authentication. A Servlet may retrieve the value of this attribute by calling the HttpServletRequest.getAuthType method. This attribute name is org.osgi.service.http.authentication.type.
   - REMOTE_USER - Specifies the name of the authenticated user. A Servlet may retrieve the value of this attribute by calling the HttpServletRequest.getRemoteUser method. This attribute name is org.osgi.service.http.authentication.remote.user.
   - AUTHORIZATION - If a User Admin service is available in the environment, then the handleSecurity method should set this attribute with the Authorization object obtained from the User Admin service. Such an object encapsulates the authentication of its remote user. A Servlet may retrieve the value of this attribute by calling ServletRequest.getAttribute(HttpContext.AUTHORIZATION). This header name is org.osgi.service.useradmin.authorization.
5. Once the request is authenticated and any attributes are set, the handleSecurity method should return true. This return indicates to the Http Service that the request is authorized and processing may continue. If the request is for a Servlet, the Http Service must then call the service method on the Servlet object.

102.8 Security

This section only applies when executing in an OSGi environment which is enforcing Java permissions.

102.8.1 Accessing Resources in Bundles

The Http Service must be granted AdminPermission[*,RESOURCE] so that bundles may use a default HttpContext object. This is necessary because the implementation of the default HttpContext object must call Bundle.getResource to access the resources of a bundle and this method requires the caller to have AdminPermission[bundle,RESOURCE].

Any bundle may access resources in its own bundle by calling Class.getResource. This operation is privileged. The resulting URL object may then be passed to the Http Service as the result of a HttpContext.getResource call. No further permission checks are performed when accessing bundle resource URL objects, so the Http Service does not need to be granted any additional permissions.

102.8.2 Accessing Other Types of Resources

In order to access resources that were not registered using the default HttpContext object, the Http Service must be granted sufficient privileges to access these resources. For example, if the getResource method of the registered HttpContext object returns a file URL, the Http Service requires the corresponding FilePermission to read the file. Similarly, if the getResource method of the registered HttpContext object returns an HTTP URL, the Http Service requires the corresponding SocketPermission to connect to the resource.

Therefore, in most cases, the Http Service should be a privileged service that is granted sufficient permission to serve any bundle's resources, no matter where these resources are located. Therefore, the Http Service must capture the AccessControlContext object of the bundle registering resources or a servlet, and then use the captured AccessControlContext object when accessing resources returned by the registered HttpContext object. This situation prevents a bundle from registering resources that it does not have permission to access.

Therefore, the Http Service should follow a scheme like the following example. When a resource or servlet is registered, it should capture the context.

```
AccessControlContext acc =
     AccessController.getContext();
```

When a URL returned by the getResource method of the associated HttpContext object is called, the Http Service must call the getResource method in a doPrivileged construct using the AccessControlContext object of the registering bundle:

```
AccessController.doPrivileged(
   new PrivilegedExceptionAction() {
      public Object run() throws Exception {
      ...
      }
   }, acc);
```

The Http Service must only use the captured AccessControlContext when accessing resource URL objects. Servlet and HttpContext objects must use a doPrivileged construct in their implementations when performing privileged operations.

102.9 Configuration Properties

If the Http Service does not have its port values configured through some other means, the Http Service implementation should use the following properties to determine the port values upon which to listen.

The following OSGi environment properties are used to specify default HTTP ports:

- org.osgi.service.http.port – This property specifies the port used for servlets and resources accessible via HTTP. The default value for this property is 80.
- org.osgi.service.http.port.secure – This property specifies the port used for servlets and resources accessible via HTTPS. The default value for this property is 443.

102.10 org.osgi.service.http

Http Service Package Version 1.2.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.http; version="[1.2,2.0)"
```

102.10.1 Summary

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

102.10.2 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.

102.10.2.1 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

102.10.2.2 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

102.10.2.3 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

102.10.2.4 public String getMimeType(String name)

name determine the MIME type for this 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.

Returns MIME type (e.g. text/html) of the name or null to indicate that the Http Service should determine the MIME type itself.

102.10.2.5 public URL getResource(String name)

name the name of the requested resource

- 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)

Returns URL that Http Service can use to read the resource or null if the resource does not exist.

102.10.2.6 public boolean handleSecurity(HttpServletRequest request, HttpServletResponse response) throws IOException

request the HTTP request

response the HTTP response

- 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.

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.

102.10.3 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

102.10.3.1 public 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

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

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.

- 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).

Throws NamespaceException – if the registration fails because the alias is already in use.

IllegalArgumentException – if any of the parameters are invalid

102.10.3.3 public void registerServlet(String alias, Servlet servlet, Dictionary initparams, HttpContext context) throws ServletException, NamespaceException

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.

- 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).

Throws NamespaceException – if the registration fails because the alias is already in use.

javax.servlet.ServletException – if the servlet's init method throws an exception, or the given servlet object has already been registered at a different alias.

IllegalArgumentException – if any of the arguments are invalid

102.10.3.4 public void unregister(String alias)

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

- 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 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 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.

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

102.10.4 public class NamespaceException extends Exception

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.

102.10.4.1 public NamespaceException(String message)

message the detail message

- Construct a NamespaceException object with a detail message.

102.10.4.2 public NamespaceException(String message, Throwable cause)

message The detail message.

cause The nested exception.

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

102.10.4.3 public Throwable getCause( )

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

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

Since 1.2

102.10.4.4 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().

102.10.4.5 public Throwable initCause(Throwable cause)

cause The cause of this exception.

- Initializes the cause of this exception to the specified value.

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

102.11 References

[1] *HTTP 1.0 Specification RFC-1945*
http://www.ietf.org/rfc/rfc1945.txt, May 1996

[2] *HTTP 1.1 Specification RFC-2616*
http://www.ietf.org/rfc/rfc2616.txt, June 1999

[3] *Java Servlet Technology*
http://java.sun.com/products/servlet/index.html

[4] *MIME Multipurpose Internet Mail Extension*
http://www.mhonarc.org/~ehood/MIME/MIME.html

[5] *Assigned MIME Media Types*
http://www.iana.org/assignments/media-types

[6] *Registration Procedures for new MIME media types*
http://www.ietf.org/rfc/rfc2048.txt

[7] *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*
http://www.ietf.org/rfc/rfc2617.txt