

FUSETM Services Framework

Using the HTTP Transport

Version 2.1.x
May 2008

Using the HTTP Transport

Version 2.1.x

Published 22 Jan 2009

Copyright © 2008 IONA Technologies PLC, a wholly-owned subsidiary of Progress Software Corporation.

Legal Notices

Progress Software Corporation and/or its subsidiaries may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this publication. Except as expressly provided in any written license agreement from Progress Software Corporation, the furnishing of this publication does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Any rights not expressly granted herein are reserved.

Progress, IONA, IONA Technologies, the IONA logo, Orbix, High Performance Integration, Artix, FUSE, and Making Software Work Together are trademarks or registered trademarks of Progress Software Corporation and/or its subsidiaries in the US and other countries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the US and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate Progress Software Corporation makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Progress Software Corporation shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third-party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this publication. This publication and features described herein are subject to change without notice. Portions of this document may include Apache Foundation documentation, all rights reserved.

Table of Contents

Basic Endpoint Addressing	11
HTTP Consumer Endpoint Configuration	13
Using Configuration	14
Using WSDL	19
Consumer Cache Control Directives	20
HTTP Service Provider Configuration	21
Using Configuration	22
Using WSDL	26
Service Provider Cache Control Directives	27
Configuring the Jetty Runtime	29
Using the Decoupled HTTP Transport	33
Index	39

List of Figures

1. Message Flow in for a Decoupled HTTP Transport	36
---	----

List of Tables

1. Elements Used to Configure an HTTP Consumer Endpoint	15
2. HTTP Consumer Configuration Attributes	15
3. http-conf:client Cache Control Directives	20
4. Elements Used to Configure an HTTP Service Provider Endpoint	23
5. HTTP Service Provider Configuration Attributes	23
6. http-conf:server Cache Control Directives	27
7. Elements for Configuring a Jetty Runtime Factory	30
8. Elements for Configuring a Jetty Runtime Instance	31
9. Attributes for Configuring a Jetty Thread Pool	31

List of Examples

1. SOAP 1.1 Port Element	11
2. SOAP 1.2 Port Element	12
3. HTTP Port Element	12
4. HTTP Consumer Configuration Namespace	14
5. http-conf:conduit Element	14
6. HTTP Consumer Endpoint Configuration	18
7. HTTP Consumer WSDL Element's Namespace	19
8. WSDL to Configure an HTTP Consumer Endpoint	19
9. HTTP Provider Configuration Namespace	22
10. http-conf:destination Element	22
11. HTTP Service Provider Endpoint Configuration	24
12. HTTP Provider WSDL Element's Namespace	26
13. WSDL to Configure an HTTP Service Provider Endpoint	26
14. Jetty Runtime Configuration Namespace	29
15. Configuring a Jetty Instance	32
16. Activating WS-Addressing using WSDL	34
17. Activating WS-Addressing using a Policy	34
18. Configuring a Consumer to Use a Decoupled HTTP Endpoint	35

Basic Endpoint Addressing

The WSDL element used to specify the address of an HTTP endpoint varies depending on the type of payload being sent over the wire.

Overview

There are three ways of specifying an HTTP endpoint's address depending on the payload format you are using.

- SOAP 1.1 uses the standardized `soap:address` element.
- SOAP 1.2 uses the `soap12:address` element.
- All other payload formats use the `http:address` element.

SOAP 1.1

When you are sending SOAP 1.1 messages over HTTP you must use the SOAP 1.1 `address` element to specify the endpoint's address. It has one attribute, `location`, that specifies the endpoint's address as a URL. The SOAP 1.1 `address` element is defined in the namespace

`http://schemas.xmlsoap.org/wsdl/soap/`.

[Example 1 on page 11](#) shows a `port` element used to send SOAP 1.1 messages over HTTP.

Example 1. SOAP 1.1 Port Element

```
<definitions ...
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" ...>
  ...
  <service name="SOAP11Service">
    <port binding="SOAP11Binding" name="SOAP11Port">
      <soap:address location="http://artie.com/index.xml">
    </port>
  </service>
  ...
</definitions>
```

SOAP 1.2

When you are sending SOAP 1.2 messages over HTTP you must use the SOAP 1.2 `address` element to specify the endpoint's address. It has one

attribute, `location`, that specifies the endpoint's address as a URL. The SOAP 1.2 `address` element is defined in the namespace

`http://schemas.xmlsoap.org/wsdl/soap12/`.

[Example 2 on page 12](#) shows a `port` element used to send SOAP 1.2 messages over HTTP.

Example 2. SOAP 1.2 Port Element

```
<definitions ...
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" ... >
  <service name="SOAP12Service">
    <port binding="SOAP12Binding" name="SOAP12Port">
      <soap12:address location="http://artie.com/index.xml">
    </port>
  </service>
  ...
</definitions>
```

Other messages types

When your messages are mapped to any payload format other than SOAP you must use the HTTP `address` element to specify the endpoint's address. It has one attribute, `location`, that specifies the endpoint's address as a URL. The HTTP `address` element is defined in the namespace

`http://schemas.xmlsoap.org/wsdl/http/`.

[Example 3 on page 12](#) shows a `port` element used to send an XML message.

Example 3. HTTP Port Element

```
<definitions ...
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" ... >
  <service name="HTTPService">
    <port binding="HTTPBinding" name="HTTPPort">
      <http:address location="http://artie.com/index.xml">
    </port>
  </service>
  ...
</definitions>
```

HTTP Consumer Endpoint Configuration

An HTTP consumer endpoint's connection properties can be configured using a configuration file or by specifying the connection properties in its WSDL document. Additional security information can be defined in the configuration file.

Using Configuration	14
Using WSDL	19
Consumer Cache Control Directives	20

HTTP consumer endpoints can specify a number of HTTP connection attributes including whether the endpoint automatically accepts redirect responses, whether the endpoint can use chunking, whether the endpoint will request a keep-alive, and how the endpoint interacts with proxies. In addition to the HTTP connection properties, an HTTP consumer endpoint can specify how it is secured.

A consumer endpoint can be configured using two mechanisms:

- [Configuration](#)
- [WSDL](#)

Using Configuration

Namespace

The elements used to configure an HTTP consumer endpoint are defined in the namespace `http://cxf.apache.org/transport/http/configuration`. It is commonly referred to using the prefix `http-conf`. In order to use the HTTP configuration elements you must add the lines shown in [Example 4 on page 14](#) to the `beans` element of your endpoint's configuration file. In addition, you must add the configuration elements' namespace to the `xsi:schemaLocation` attribute.

Example 4. HTTP Consumer Configuration Namespace

```
<beans ...
  xmlns:http-conf="http://cxf.apache.org/transport/http/configuration
  ...
  xsi:schemaLocation="...
    http://cxf.apache.org/transport/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
  ...>
```

The conduit element

You configure an HTTP endpoint using the `http-conf:conduit` element and its children. The `http-conf:conduit` element takes a single attribute, `name`, that specifies the WSDL `port` element corresponding to the endpoint. The value for the `name` attribute takes the form `portQName.http-conduit`. [Example 5 on page 14](#) shows the `http-conf:conduit` element that would be used to add configuration for an endpoint that is specified by the WSDL fragment `<port binding="widgetSOAPBinding" name="widgetSOAPPort">` when the endpoint's target namespace is `http://widgets.widgetvendor.net`.

Example 5. http-conf:conduit Element

```
...
<http-conf:conduit name="{http://widgets/widgetvendor.net}widgetSOAPPort.http-conduit"
  ...
</http-conf:conduit>
...
```

The `http-conf:conduit` element has child elements that specify configuration information. They are described in [Table 1 on page 15](#).

Table 1. Elements Used to Configure an HTTP Consumer Endpoint

Element	Description
<code>http-conf:client</code>	Specifies the HTTP connection properties such as timeouts, keep-alive requests, content types, etc. See The client element on page 15 .
<code>http-conf:authorization</code>	Specifies the parameters for configuring the basic authentication method that the endpoint uses preemptively. The preferred approach is to supply a Basic Authentication Supplier object.
<code>http-conf:proxyAuthorization</code>	Specifies the parameters for configuring basic authentication against outgoing HTTP proxy servers.
<code>http-conf:tlsClientParameters</code>	Specifies the parameters used to configure SSL/TLS.
<code>http-conf:basicAuthSupplier</code>	Specifies the bean reference or class name of the object that supplies the basic authentication information used by the endpoint, either preemptively or in response to a 401 HTTP challenge.
<code>http-conf:trustDecider</code>	Specifies the bean reference or class name of the object that checks the HTTP(S) <code>URLConnection</code> object to establish trust for a connection with an HTTPS service provider before any information is transmitted.

The client element

The `http-conf:client` element is used to configure the non-security properties of a consumer endpoint's HTTP connection. Its attributes, described in [Table 2 on page 15](#), specify the connection's properties.

Table 2. HTTP Consumer Configuration Attributes

Attribute	Description
<code>ConnectionTimeout</code>	Specifies the amount of time, in milliseconds, that the consumer attempts to establish a connection before it times out. The default is 30000. 0 specifies that the consumer will continue to send the request indefinitely.
<code>ReceiveTimeout</code>	Specifies the amount of time, in milliseconds, that the consumer will wait for a response before it times out. The default is 30000. 0 specifies that the consumer will wait indefinitely.

Attribute	Description
AutoRedirect	Specifies if the consumer will automatically follow a server issued redirection. The default is <code>false</code> .
MaxRetransmits	Specifies the maximum number of times a consumer will retransmit a request to satisfy a redirect. The default is <code>-1</code> which specifies that unlimited retransmissions are allowed.
AllowChunking	<p>Specifies whether the consumer will send requests using chunking. The default is <code>true</code> which specifies that the consumer will use chunking when sending requests.</p> <p>Chunking cannot be used if either of the following are true:</p> <ul style="list-style-type: none"> • <code>http-conf:basicAuthSupplier</code> is configured to provide credentials preemptively. • <code>AutoRedirect</code> is set to <code>true</code>. <p>In both cases the value of <code>AllowChunking</code> is ignored and chunking is disallowed.</p>
Accept	Specifies what media types the consumer is prepared to handle. The value is used as the value of the HTTP Accept property. The value of the attribute is specified using multipurpose internet mail extensions (MIME) types.
AcceptLanguage	<p>Specifies what language (for example, American English) the consumer prefers for the purpose of receiving a response. The value is used as the value of the HTTP AcceptLanguage property.</p> <p>Language tags are regulated by the International Organization for Standards (ISO) and are typically formed by combining a language code, determined by the ISO-639 standard, and country code, determined by the ISO-3166 standard, separated by a hyphen. For example, <code>en-US</code> represents American English.</p>
AcceptEncoding	Specifies what content encodings the consumer is prepared to handle. Content encoding labels are regulated by the Internet Assigned Numbers Authority (IANA). The value is used as the value of the HTTP AcceptEncoding property.
ContentType	<p>Specifies the media type of the data being sent in the body of a message. Media types are specified using multipurpose internet mail extensions (MIME) types. The value is used as the value of the HTTP ContentType property. The default is <code>text/xml</code>.</p> <p>For web services, this should be set to <code>text/xml</code>. If the client is sending HTML form data to a CGI script, this should be set to <code>application/x-www-form-urlencoded</code>. If the HTTP POST request is bound to a fixed payload format (as opposed to SOAP), the content type is typically set to <code>application/octet-stream</code>.</p>

Attribute	Description
Host	<p>Specifies the Internet host and port number of the resource on which the request is being invoked. The value is used as the value of the HTTP Host property.</p> <p>This attribute is typically not required. It is only required by certain DNS scenarios or application designs. For example, it indicates what host the client prefers for clusters (that is, for virtual servers mapping to the same Internet protocol (IP) address).</p>
Connection	<p>Specifies whether a particular connection is to be kept open or closed after each request/response dialog. There are two valid values:</p> <ul style="list-style-type: none"> • <code>Keep-Alive</code> — Specifies that the consumer wants the connection kept open after the initial request/response sequence. If the server honors it, the connection is kept open until the consumer closes it. • <code>close(default)</code> — Specifies that the connection to the server is closed after each request/response sequence.
CacheControl	<p>Specifies directives about the behavior that must be adhered to by caches involved in the chain comprising a request from a consumer to a service provider. See Consumer Cache Control Directives on page 20.</p>
Cookie	<p>Specifies a static cookie to be sent with all requests.</p>
BrowserType	<p>Specifies information about the browser from which the request originates. In the HTTP specification from the World Wide Web consortium (W3C) this is also known as the <i>user-agent</i>. Some servers optimize based on the client that is sending the request.</p>
Referer	<p>Specifies the URL of the resource that directed the consumer to make requests on a particular service. The value is used as the value of the HTTP Referer property.</p> <p>This HTTP property is used when a request is the result of a browser user clicking on a hyperlink rather than typing a URL. This can allow the server to optimize processing based upon previous task flow, and to generate lists of back-links to resources for the purposes of logging, optimized caching, tracing of obsolete or mistyped links, and so on. However, it is typically not used in web services applications.</p> <p>If the <code>AutoRedirect</code> attribute is set to <code>true</code> and the request is redirected, any value specified in the <code>Referer</code> attribute is overridden. The value of the HTTP Referer property is set to the URL of the service that redirected the consumer's original request.</p>
DecoupledEndpoint	<p>Specifies the URL of a decoupled endpoint for the receipt of responses over a separate provider->consumer connection. For more information on using decoupled endpoints see, Using the Decoupled HTTP Transport on page 33.</p>

Attribute	Description
	You must configure both the consumer endpoint and the service provider endpoint to use WS-Addressing for the decoupled endpoint to work.
ProxyServer	Specifies the URL of the proxy server through which requests are routed.
ProxyServerPort	Specifies the port number of the proxy server through which requests are routed.
ProxyServerType	Specifies the type of proxy server used to route requests. Valid values are: <ul style="list-style-type: none"> • HTTP(default) • SOCKS

Example

[Example 6 on page 18](#) shows the configuration of an HTTP consumer endpoint that wants to keep its connection to the provider open between requests, that will only retransmit requests once per invocation, and that cannot use chunking streams.

Example 6. HTTP Consumer Endpoint Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:http-conf="http://cxf.apache.org/transports/http/configuration"
  xsi:schemaLocation="http://cxf.apache.org/transports/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

  <http-conf:conduit name="{http://apache.org/hello_world_soap_http}SoapPort.http-conduit">

    <http-conf:client Connection="Keep-Alive"
      MaxRetransmits="1"
      AllowChunking="false" />
  </http-conf:conduit>
</beans>
```

Using WSDL

Namespace

The WSDL extension elements used to configure an HTTP consumer endpoint are defined in the namespace

`http://cxf.apache.org/transports/http/configuration`. It is commonly referred to using the prefix `http-conf`. In order to use the HTTP configuration elements you must add the line shown in [Example 7 on page 19](#) to the `definitions` element of your endpoint's WSDL document.

Example 7. HTTP Consumer WSDL Element's Namespace

```
<definitions ...  
  xmlns:http-conf="http://cxf.apache.org/transports/http/configuration
```

The client element

The `http-conf:client` element is used to specify the connection properties of an HTTP consumer in a WSDL document. The `http-conf:client` element is a child of the WSDL `port` element. It has the same attributes as the `client` element used in the configuration file. The attributes are described in [Table 2 on page 15](#).

Example

[Example 8 on page 19](#) shows a WSDL fragment that configures an HTTP consumer endpoint to specify that it does not interact with caches.

Example 8. WSDL to Configure an HTTP Consumer Endpoint

```
<service ...>  
  <port ...>  
    <soap:address ... />  
    <http-conf:client CacheControl="no-cache" />  
  </port>  
</service>
```

Consumer Cache Control Directives

Table 3 on page 20 lists the cache control directives supported by an HTTP consumer.

Table 3. *http-conf:client Cache Control Directives*

Directive	Behavior
no-cache	Caches cannot use a particular response to satisfy subsequent requests without first revalidating that response with the server. If specific response header fields are specified with this value, the restriction applies only to those header fields within the response. If no response header fields are specified, the restriction applies to the entire response.
no-store	Caches must not store either any part of a response or any part of the request that invoked it.
max-age	The consumer can accept a response whose age is no greater than the specified time in seconds.
max-stale	The consumer can accept a response that has exceeded its expiration time. If a value is assigned to max-stale, it represents the number of seconds beyond the expiration time of a response up to which the consumer can still accept that response. If no value is assigned, the consumer can accept a stale response of any age.
min-fresh	The consumer wants a response that is still fresh for at least the specified number of seconds indicated.
no-transform	Caches must not modify media type or location of the content in a response between a provider and a consumer.
only-if-cached	Caches should return only responses that are currently stored in the cache, and not responses that need to be reloaded or revalidated.
cache-extension	Specifies additional extensions to the other cache directives. Extensions can be informational or behavioral. An extended directive is specified in the context of a standard directive, so that applications not understanding the extended directive can adhere to the behavior mandated by the standard directive.

HTTP Service Provider Configuration

An HTTP service provider's connection properties can be configured using a configuration file or by specifying the connection properties in its WSDL document.

Using Configuration	22
Using WSDL	26
Service Provider Cache Control Directives	27

HTTP service provider endpoints can specify a number of HTTP connection attributes including if it will honor keep alive requests, how it interacts with caches, and how tolerant it is of errors in communicating with a consumer.

A service provider endpoint can be configured using two mechanisms:

- [Configuration](#)
- [WSDL](#)

Using Configuration

Namespace

The elements used to configure an HTTP provider endpoint are defined in the namespace `http://cxf.apache.org/transport/http/configuration`. It is commonly referred to using the prefix `http-conf`. In order to use the HTTP configuration elements you must add the lines shown in [Example 9 on page 22](#) to the `beans` element of your endpoint's configuration file. In addition, you must add the configuration elements' namespace to the `xsi:schemaLocation` attribute.

Example 9. HTTP Provider Configuration Namespace

```
<beans ...
  xmlns:http-conf="http://cxf.apache.org/transport/http/configuration
  ...
  xsi:schemaLocation="...
    http://cxf.apache.org/transport/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
  ...>
```

The destination element

You configure an HTTP service provider endpoint using the `http-conf:destination` element and its children. The `http-conf:destination` element takes a single attribute, `name`, that specifies the WSDL `port` element that corresponds to the endpoint. The value for the `name` attribute takes the form `portQName.http-destination`.

[Example 10 on page 22](#) shows the `http-conf:destination` element that is used to add configuration for an endpoint that is specified by the WSDL fragment `<port binding="widgetSOAPBinding" name="widgetSOAPPort">` when the endpoint's target namespace is `http://widgets.widgetvendor.net`.

Example 10. `http-conf:destination` Element

```
...
<http-conf:destination name="{http://widgets/widgetvendor.net}widgetSOAPPort.http-destination">
  ...
</http-conf:destination>
...
```

The `http-conf:destination` element has a number of child elements that specify configuration information. They are described in [Table 4 on page 23](#).

Table 4. Elements Used to Configure an HTTP Service Provider Endpoint

Element	Description
<code>http-conf:server</code>	Specifies the HTTP connection properties. See The server element on page 23 .
<code>http-conf:contextMatchStrategy</code>	Specifies the parameters that configure the context match strategy for processing HTTP requests.
<code>http-conf:fixedParameterOrder</code>	Specifies whether the parameter order of an HTTP request handled by this destination is fixed.

The server element

The `http-conf:server` element is used to configure the properties of a service provider endpoint's HTTP connection. Its attributes, described in [Table 5 on page 23](#), specify the connection's properties.

Table 5. HTTP Service Provider Configuration Attributes

Attribute	Description
<code>ReceiveTimeout</code>	Sets the length of time, in milliseconds, the service provider attempts to receive a request before the connection times out. The default is <code>30000</code> . <code>0</code> specifies that the provider will not timeout.
<code>SuppressClientSendErrors</code>	Specifies whether exceptions are to be thrown when an error is encountered on receiving a request. The default is <code>false</code> ; exceptions are thrown on encountering errors.
<code>SuppressClientReceiveErrors</code>	Specifies whether exceptions are to be thrown when an error is encountered on sending a response to a consumer. The default is <code>false</code> ; exceptions are thrown on encountering errors.
<code>HonorKeepAlive</code>	Specifies whether the service provider honors requests for a connection to remain open after a response has been sent. The default is <code>false</code> ; keep-alive requests are ignored.
<code>RedirectURL</code>	Specifies the URL to which the client request should be redirected if the URL specified in the client request is no longer appropriate for the requested resource. In this case, if a status code is not automatically set in the first line of the server response, the status code is set to <code>302</code> and the status description

Attribute	Description
	is set to <code>Object Moved</code> . The value is used as the value of the HTTP <code>RedirectURL</code> property.
<code>CacheControl</code>	Specifies directives about the behavior that must be adhered to by caches involved in the chain comprising a response from a service provider to a consumer. See Service Provider Cache Control Directives on page 27 .
<code>ContentLocation</code>	Sets the URL where the resource being sent in a response is located.
<code>ContentType</code>	Specifies the media type of the information being sent in a response. Media types are specified using multipurpose internet mail extensions (MIME) types. The value is used as the value of the HTTP <code>ContentType</code> location.
<code>ContentEncoding</code>	<p>Specifies any additional content encodings that have been applied to the information being sent by the service provider. Content encoding labels are regulated by the Internet Assigned Numbers Authority (IANA). Possible content encoding values include <code>zip</code>, <code>gzip</code>, <code>compress</code>, <code>deflate</code>, and <code>identity</code>.</p> <p>This value is used as the value of the HTTP <code>ContentEncoding</code> property.</p> <p>The primary use of content encodings is to allow documents to be compressed using some encoding mechanism, such as <code>zip</code> or <code>gzip</code>. FUSE Services Framework performs no validation on content codings. It is the user's responsibility to ensure that a specified content coding is supported at application level.</p>
<code>ServerType</code>	Specifies what type of server is sending the response. Values take the form <code>program-name/version</code> ; for example, <code>Apache/1.2.5</code> .

Example

[Example 11 on page 24](#) shows the configuration for an HTTP service provider endpoint that honors keep-alive requests and suppresses all communication errors.

Example 11. HTTP Service Provider Endpoint Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:http-conf="http://cxf.apache.org/transports/http/configuration"
  xsi:schemaLocation="http://cxf.apache.org/transports/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

  <http-conf:destination name="{http://apache.org/hello_world_soap_http}SoapPort.http-des
```

```
    tination">
      <http-conf:server SuppressClientSendErrors="true"
        SuppressClientReceiveErrors="true"
        HonorKeepAlive="true" />
    </http-conf:destination>
</beans>
```

Using WSDL

Namespace

The WSDL extension elements used to configure an HTTP provider endpoint are defined in the namespace

`http://cxf.apache.org/transports/http/configuration`. It is commonly referred to using the prefix `http-conf`. To use the HTTP configuration elements you must add the line shown in [Example 12 on page 26](#) to the `definitions` element of your endpoint's WSDL document.

Example 12. HTTP Provider WSDL Element's Namespace

```
<definitions ...  
  xmlns:http-conf="http://cxf.apache.org/transports/http/configuration
```

The server element

The `http-conf:server` element is used to specify the connection properties of an HTTP service provider in a WSDL document. The `http-conf:server` element is a child of the WSDL `port` element. It has the same attributes as the `server` element used in the configuration file. The attributes are described in [Table 5 on page 23](#).

Example

[Example 13 on page 26](#) shows a WSDL fragment that configures an HTTP service provider endpoint specifying that it will not interact with caches.

Example 13. WSDL to Configure an HTTP Service Provider Endpoint

```
<service ...>  
  <port ...>  
    <soap:address ... />  
    <http-conf:server CacheControl="no-cache" />  
  </port>  
</service>
```

Service Provider Cache Control Directives

Table 6 on page 27 lists the cache control directives supported by an HTTP service provider.

Table 6. *http-conf:server Cache Control Directives*

Directive	Behavior
no-cache	Caches cannot use a particular response to satisfy subsequent requests without first revalidating that response with the server. If specific response header fields are specified with this value, the restriction applies only to those header fields within the response. If no response header fields are specified, the restriction applies to the entire response.
public	Any cache can store the response.
private	Public (<i>shared</i>) caches cannot store the response because the response is intended for a single user. If specific response header fields are specified with this value, the restriction applies only to those header fields within the response. If no response header fields are specified, the restriction applies to the entire response.
no-store	Caches must not store any part of the response or any part of the request that invoked it.
no-transform	Caches must not modify the media type or location of the content in a response between a server and a client.
must-revalidate	Caches must revalidate expired entries that relate to a response before that entry can be used in a subsequent response.
proxy-revalidate	Does the same as must-revalidate, except that it can only be enforced on shared caches and is ignored by private unshared caches. When using this directive, the public cache directive must also be used.
max-age	Clients can accept a response whose age is no greater than the specified number of seconds.
s-max-age	Does the same as max-age, except that it can only be enforced on shared caches and is ignored by private unshared caches. The age specified by s-max-age overrides the age specified by max-age. When using this directive, the proxy-revalidate directive must also be used.
cache-extension	Specifies additional extensions to the other cache directives. Extensions can be informational or behavioral. An extended directive is specified in the context of a standard directive, so that applications not understanding the extended directive can adhere to the behavior mandated by the standard directive.

Configuring the Jetty Runtime

The Jetty instance used to implement the HTTP server runtime has a number of configurable properties.

Overview

The Jetty runtime is used by HTTP service providers and HTTP consumers using a decoupled endpoint. The runtime's thread pool can be configured, and you can also set a number of the security settings for an HTTP service provider through the Jetty runtime.

Namespace

The elements used to configure the Jetty runtime are defined in the namespace `http://cxf.apache.org/transport/http-jetty/configuration`. It is commonly referred to using the prefix `httpj`. In order to use the Jetty configuration elements you must add the lines shown in [Example 14 on page 29](#) to the `beans` element of your endpoint's configuration file. In addition, you must add the configuration elements' namespace to the `xsi:schemaLocation` attribute.

Example 14. Jetty Runtime Configuration Namespace

```
<beans ...
  xmlns:httpj="http://cxf.apache.org/transport/http-jetty/configuration
  ...
  xsi:schemaLocation="...
    http://cxf.apache.org/transport/http-jetty/configuration
    http://cxf.apache.org/schemas/configuration/http-jetty.xsd
  ...>
```

The engine-factory element

The `httpj:engine-factory` element is the root element used to configure the Jetty runtime used by an application. It has a single required attribute, `bus`, whose value is the name of the `Bus` that manages the Jetty instances being configured.



Tip

The value is typically `cxf` which is the name of the default `Bus` instance.

The `httpj:engine-factory` element has three children that contain the information used to configure the HTTP ports instantiated by the Jetty runtime factory. The children are described in [Table 7 on page 30](#).

Table 7. Elements for Configuring a Jetty Runtime Factory

Element	Description
<code>httpj:engine</code>	Specifies the configuration for a particular Jetty runtime instance. See The engine element on page 30 .
<code>httpj:identifiedTLSServerParameters</code>	Specifies a reusable set of properties for securing an HTTP service provider. It has a single attribute, <code>id</code> , that specifies a unique identifier by which the property set can be referred.
<code>httpj:identifiedThreadingParameters</code>	Specifies a reusable set of properties for controlling a Jetty instance's thread pool. It has a single attribute, <code>id</code> , that specifies a unique identifier by which the property set can be referred. See Configuring the thread pool on page 31 .

The engine element

The `httpj:engine` element is used to configure specific instances of the Jetty runtime. It has a single attribute, `port`, that specifies the number of the port being managed by the Jetty instance.



Tip

You can specify a value of 0 for the `port` attribute. Any threading properties specified in an `httpj:engine` element with its `port` attribute set to 0 are used as the configuration for all Jetty listeners that are not explicitly configured.

Each `httpj:engine` element can have two children: one for configuring security properties and one for configuring the Jetty instance's thread pool. For each type of configuration you can either directly provide the configuration information or you can provide a reference to a set of configuration properties defined in the parent `httpj:engine-factory` element.

The child elements used to provide the configuration properties are described in [Table 8 on page 31](#).

Table 8. Elements for Configuring a Jetty Runtime Instance

Element	Description
<code>httpj:tlsServerParameters</code>	Specifies a set of properties for configuring the security used for the specific Jetty instance.
<code>httpj:tlsServerParametersRef</code>	Refers to a set of security properties defined by a <code>identifiedTLSServerParameters</code> element. The <code>id</code> attribute provides the id of the referred <code>identifiedTLSServerParameters</code> element.
<code>httpj:threadingParameters</code>	Specifies the size of the thread pool used by the specific Jetty instance. See Configuring the thread pool on page 31 .
<code>httpj:threadingParametersRef</code>	Refers to a set of properties defined by a <code>identifiedThreadingParameters</code> element. The <code>id</code> attribute provides the id of the referred <code>identifiedThreadingParameters</code> element.

Configuring the thread pool

You can configure the size of a Jetty instance's thread pool by either:

- Specifying the size of the thread pool using a `identifiedThreadingParameters` element in the `engine-factory` element. You then refer to the element using a `threadingParametersRef` element.
- Specifying the size of the of the thread pool directly using a `threadingParameters` element.

The `threadingParameters` has two attributes to specify the size of a thread pool. The attributes are described in [Table 9 on page 31](#).



Note

The `httpj:identifiedThreadingParameters` element has a single child `threadingParameters` element.

Table 9. Attributes for Configuring a Jetty Thread Pool

Attribute	Description
<code>minThreads</code>	Specifies the minimum number of threads available to the Jetty instance for processing requests.

Attribute	Description
maxThreads	Specifies the maximum number of threads available to the Jetty instance for processing requests.

Example

[Example 15 on page 32](#) shows a configuration fragment that configures a Jetty instance on port number 9001.

Example 15. Configuring a Jetty Instance

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sec="http://cxf.apache.org/configuration/security"
  xmlns:http="http://cxf.apache.org/transports/http/configuration"
  xmlns:httpj="http://cxf.apache.org/transports/http-jetty/configuration"
  xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
  xsi:schemaLocation="http://cxf.apache.org/configuration/security
    http://cxf.apache.org/schemas/configuration/security.xsd
    http://cxf.apache.org/transports/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
    http://cxf.apache.org/transports/http-jetty/configuration
    http://cxf.apache.org/schemas/configuration/http-jetty.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
  ...

  <httpj:engine-factory bus="cxf">
    <httpj:identifiedTLSServerParameters id="secure">
      <sec:keyManagers keyPassword="password">
        <sec:keyStore type="JKS" password="password"
          file="certs/cherry.jks"/>
      </sec:keyManagers>
    </httpj:identifiedTLSServerParameters>

    <httpj:engine port="9001">
      <httpj:tlsServerParametersRef id="secure" />
      <httpj:threadingParameters minThreads="5"
        maxThreads="15" />
    </httpj:engine>
  </httpj:engine-factory>
</beans>
```

Using the Decoupled HTTP Transport

The decoupled HTTP transport allows you to send requests and responses using separate HTTP connections.

Overview

In normal HTTP request/response scenarios, the request and the response are sent using the same HTTP connection. The service provider processes the request and responds with a response containing the appropriate HTTP status code and the contents of the response. In the case of a successful request, the HTTP status code is set to 200.

In some instances, such as when using WS-RM or when requests take an extended period of time to execute, it makes sense to decouple the request and response message. In this case the service providers sends the consumer a 202 `Accepted` response to the consumer over the back-channel of the HTTP connection on which the request was received. It then processes the request and sends the response back to the consumer using a new decoupled server->client HTTP connection. The consumer runtime receives the incoming response and correlates it with the appropriate request before returning to the application code.

Configuring decoupled interactions

Using the HTTP transport in decoupled mode requires that you do the following:

1. Configure the consumer to use WS-Addressing.
See [Configuring an endpoint to use WS-Addressing on page 33](#).
 2. Configure the consumer to use a decoupled endpoint.
See [Configuring the consumer on page 34](#).
 3. Configure any service providers that the consumer interacts with to use WS-Addressing.
See [Configuring an endpoint to use WS-Addressing on page 33](#).
-

Configuring an endpoint to use WS-Addressing

Specify that the consumer and any service provider with which the consumer interacts use WS-Addressing.

You can specify that an endpoint uses WS-Addressing in one of two ways:

- Adding the `wsa:UsingAddressing` element to the endpoint's WSDL `port` element as shown in [Example 16 on page 34](#).

Example 16. Activating WS-Addressing using WSDL

```
...  
<service name="WidgetSOAPService">  
  <port name="WidgetSOAPPort" binding="tns:WidgetSOAPBinding">  
    <soap:address="http://widgetvendor.net/widgetSeller" />  
    <wsa:UsingAddressing xmlns:wsa="http://www.w3.org/2005/02/addressing/wsdl"/>  
  </port>  
</service>  
...
```

- Adding the WS-Addressing policy to the endpoint's WSDL `port` element as shown in [Example 17 on page 34](#).

Example 17. Activating WS-Addressing using a Policy

```
...  
<service name="WidgetSOAPService">  
  <port name="WidgetSOAPPort" binding="tns:WidgetSOAPBinding">  
    <soap:address="http://widgetvendor.net/widgetSeller" />  
    <wsp:Policy xmlns:wsp="http://www.w3.org/2006/07/ws-policy">  
      <wsam:Addressing xmlns:wsam="http://www.w3.org/2007/02/addressing/metadata">  
        <wsp:Policy/>  
      </wsam:Addressing>  
    </wsp:Policy>  
  </port>  
</service>  
...
```



Note

The WS-Addressing policy supersedes the `wsa:UsingAddressing` WSDL element.

Configuring the consumer

Configure the consumer endpoint to use a decoupled endpoint using the `DecoupledEndpoint` attribute of the `http-conf:conduit` element.

[Example 18 on page 35](#) shows the configuration for setting up the endpoint defined in [Example 16 on page 34](#) to use a decoupled endpoint. The

consumer now receives all responses at
`http://widgetvendor.net/widgetSellerInbox.`

Example 18. Configuring a Consumer to Use a Decoupled HTTP Endpoint

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:http="http://cxf.apache.org/transports/http/configuration"
  xsi:schemaLocation="http://cxf.apache.org/transports/http/configuration
    http://cxf.apache.org/schemas/configuration/http-conf.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

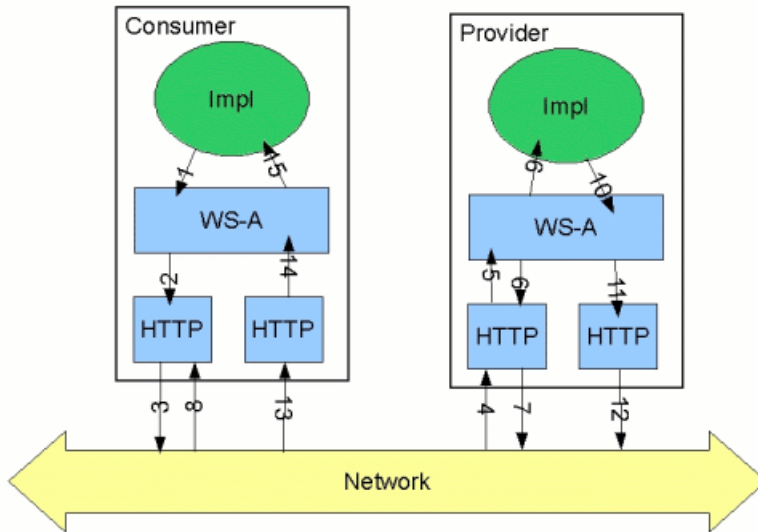
  <http:conduit name="{http://widgetvendor.net/services}WidgetSOAPPort.http-conduit">
    <http:client DecoupledEndpoint="http://widgetvendor.net:9999/decoupled_endpoint" />
  </http:conduit>
</beans>
```

How messages are processed

Using the HTTP transport in decoupled mode adds extra layers of complexity to the processing of HTTP messages. While the added complexity is transparent to the implementation level code in an application, it might be important to understand what happens for debugging reasons.

[Figure 1 on page 36](#) shows the flow of messages when using HTTP in decoupled mode.

Figure 1. Message Flow in for a Decoupled HTTP Transport



A request starts the following process:

1. The consumer implementation invokes an operation and a request message is generated.
2. The WS-Addressing layer adds the WS-A headers to the message.

When a decoupled endpoint is specified in the consumer's configuration, the address of the decoupled endpoint is placed in the WS-A ReplyTo header.

3. The message is sent to the service provider.

4. The service provider receives the message.
5. The request message from the consumer is dispatched to the provider's WS-A layer.
6. Because the WS-A ReplyTo header is not set to anonymous, the provider sends back a message with the HTTP status code set to 202, acknowledging that the request has been received.
7. The HTTP layer sends a 202 Accepted message back to the consumer using the original connection's back-channel.
8. The consumer receives the 202 Accepted reply on the back-channel of the HTTP connection used to send the original message.

When the consumer receives the 202 Accepted reply, the HTTP connection closes.

9. The request is passed to the service provider's implementation where the request is processed.
10. When the response is ready, it is dispatched to the WS-A layer.
11. The WS-A layer adds the WS-Addressing headers to the response message.
12. The HTTP transport sends the response to the consumer's decoupled endpoint.
13. The consumer's decoupled endpoint receives the response from the service provider.
14. The response is dispatched to the consumer's WS-A layer where it is correlated to the proper request using the WS-A RelatesTo header.
15. The correlated response is returned to the client implementation and the invoking call is unblocked.

Index

C

- configuration
 - HTTP consumer connection properties, 15
 - HTTP consumer endpoint, 14
 - HTTP service provider connection properties, 23
 - HTTP service provider endpoint, 22
 - HTTP thread pool, 31
 - Jetty engine, 29
 - Jetty instance, 30

H

- http-conf:authorization, 15
- http-conf:basicAuthSupplier, 15
- http-conf:client, 15
 - Accept, 16
 - AcceptEncoding, 16
 - AcceptLanguage, 16
 - AllowChunking, 16
 - AutoRedirect, 16
 - BrowserType, 17
 - CacheControl, 17, 20
 - Connection, 17
 - ConnectionTimeout, 15
 - ContentType, 16
 - Cookie, 17
 - DecoupledEndpoint, 17, 34
 - Host, 17
 - MaxRetransmits, 16
 - ProxyServer, 18
 - ProxyServerPort, 18
 - ProxyServerType, 18
 - ReceiveTimeout, 15
 - Referer, 17
- http-conf:conduit, 14
 - name attribute, 14
- http-conf:contextMatchStrategy, 23
- http-conf:destination, 22
 - name attribute, 22
- http-conf:fixedParameterOrder, 23

- http-conf:proxyAuthorization, 15
- http-conf:server, 23
 - CacheControl, 24, 27
 - ContentEncoding, 24
 - ContentLocation, 24
 - ContentType, 24
 - HonorKeepAlive, 23
 - ReceiveTimeout, 23
 - RedirectURL, 23
 - ServerType, 24
 - SuppressClientReceiveErrors, 23
 - SuppressClientSendErrors, 23
- http-conf:tlsClientParameters, 15
- http-conf:trustDecider, 15
- http:address, 12
- httpj:engine, 30
- httpj:engine-factory, 29
- httpj:identifiedThreadingParameters, 30, 31
- httpj:identifiedTLSServerParameters, 30
- httpj:threadingParameters, 31
 - maxThreads, 32
 - minThreads, 31
- httpj:threadingParametersRef, 31
- httpj:tlsServerParameters, 31
- httpj:tlsServerParametersRef, 31

S

- SOAP 1.1
 - endpoint address, 11
- SOAP 1.2
 - endpoint address, 11
- soap12:address, 11
- soap:address, 11

W

- WS-Addressing
 - using, 33
- wsam:Addressing, 33
- wswa:UsingAddressing, 33

