

## Web services

### Contents

1. [WebSphere MQ transport for SOAP](#)
  - 1.1. [Introduction](#)
    - 1.1.1. [Integration of SOAP and WebSphere MQ](#)
    - 1.1.2. [Implementation details for .NET Framework 1 and 2, and Axis 1.4](#)
    - 1.1.3. [Web services reliable messaging](#)
  - 1.2. [Installing WebSphere MQ Web transport for SOAP](#)
  - 1.3. [Verifying the WebSphere MQ transport for SOAP](#)
  - 1.4. [Developing WebSphere MQ Web services](#)
    - 1.4.1. [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)
    - 1.4.2. [Developing a .NET 1 or 2 service using Visual Studio 2008](#)
    - 1.4.3. [Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)
  - 1.5. [Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP](#)
    - 1.5.1. [Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse](#)
    - 1.5.2. [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)
    - 1.5.3. [Developing a .NET 1 or 2 client for WebSphere transport for SOAP using Microsoft Visual Studio 2008](#)
  - 1.6. [Deploying Web services using the WebSphere MQ transport for SOAP](#)
    - 1.6.1. [Deploying to Axis 1.4 using amqwdeployWMOService](#)
    - 1.6.2. [Deploying to .NET Framework 1 and 2](#)
    - 1.6.3. [Deploying to CICS Transaction Server](#)
    - 1.6.4. [Deploying to WebSphere Application Server](#)
    - 1.6.5. [Configuring WebSphere Application Server to use W3C SOAP over JMS](#)
    - 1.6.6. [Deploying to WebSphere ESB and Process Server](#)
  - 1.7. [Deploying Web service clients](#)
    - 1.7.1. [Deploying to Axis 1.4](#)
    - 1.7.2. [Deploying to Axis2](#)
    - 1.7.3. [Deploying to Axis2 using W3C SOAP over JMS](#)
    - 1.7.4. [Deploying to .NET Framework 1 and 2](#)
  - 1.8. [Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS](#)
  - 1.9. [Reference](#)
    - 1.9.1. [amqSOAPNETListener: Websphere MQ SOAP Listener for .NET Framework 1 or 2](#)
    - 1.9.2. [amqwsdl: generate WSDL for .NET service](#)
    - 1.9.3. [amqwclientconfig: create client deployment descriptor](#)
    - 1.9.4. [amqwdeployWMOService: deploy Web service utility](#)
      - 1.9.4.1. [Output files](#)
      - 1.9.4.2. [Usage notes](#)
    - 1.9.5. [amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET](#)
    - 1.9.6. [Apache software license](#)
    - 1.9.7. [MQMD SOAP settings](#)
    - 1.9.8. [MQRFH2 SOAP settings](#)
    - 1.9.9. [runivt: installation verification test](#)
    - 1.9.10. [Secure WebSphere MQ Web services](#)
      - 1.9.10.1. [SSL and the WebSphere MQ transport for SOAP](#)
      - 1.9.10.2. [SSL connection factory parameters in the WebSphere MQ Web services URI](#)
    - 1.9.11. [SimpleJavaListener: Websphere MQ SOAP Listener for Axis 1.4](#)
    - 1.9.12. [SOAP listeners](#)
    - 1.9.13. [SOAP senders](#)
      - 1.9.13.1. [Use a channel definition table](#)
    - 1.9.14. [Transactions](#)
    - 1.9.15. [WebSphere MQ transport for SOAP URI](#)
      - 1.9.15.1. [The Nojindi mechanism](#)
    - 1.9.16. [W3C SOAP over JMS URI](#)
    - 1.9.17. [WebSphere MQ transport for SOAP Web services](#)
    - 1.9.18. [WebSphere MQ transport for SOAP Web service clients](#)
2. [WebSphere MQ bridge for HTTP](#)
  - 2.1. [Introduction to WebSphere MQ bridge for HTTP](#)
  - 2.2. [Installing, configuring, and verifying](#)
    - 2.2.1. [Example: WebSphere Application Server V6.1.0.9](#)
    - 2.2.2. [Configuring WebSphere MQ Bridge for HTTP](#)
      - 2.2.2.1. [Configuring WebSphere MQ Bridge for HTTP to implement diagnostic tracing](#)
      - 2.2.2.2. [Configuring your connection factory](#)
  - 2.3. [Publish/subscribe](#)
  - 2.4. [Running the samples](#)
  - 2.5. [Security considerations](#)
  - 2.6. [Reference](#)
    - 2.6.1. [HTTP DELETE](#)
    - 2.6.2. [HTTP GET](#)
    - 2.6.3. [HTTP POST](#)
    - 2.6.4. [HTTP headers](#)
      - 2.6.4.1. [class: x-msg-class entity-header](#)
      - 2.6.4.2. [Content-Length: HTTP entity-header](#)
      - 2.6.4.3. [Content-Location: HTTP entity-header](#)
      - 2.6.4.4. [Content-Range: HTTP entity-header](#)
      - 2.6.4.5. [Content-Type: HTTP entity-header](#)
      - 2.6.4.6. [correlId: x-msg-correlId entity-header](#)
      - 2.6.4.7. [encoding: x-msg-encoding entity-header](#)
      - 2.6.4.8. [expiry: x-msg-expiry entity-header](#)
      - 2.6.4.9. [format: x-msg-format entity-header](#)
      - 2.6.4.10. [msgId: x-msg-msgId entity-header](#)
      - 2.6.4.11. [persistence: x-msg-persistence entity-header](#)
      - 2.6.4.12. [priority: x-msg-priority entity-header](#)
      - 2.6.4.13. [range: x-msg-range request-header](#)
      - 2.6.4.14. [replyTo: x-msg-replyTo entity-header](#)
      - 2.6.4.15. [Server: HTTP response-header](#)
      - 2.6.4.16. [require-headers: x-msg-require-headers request-header](#)
      - 2.6.4.17. [timestamp: x-msg-timestamp entity-header](#)
      - 2.6.4.18. [usr: x-msg-usr entity-header](#)
      - 2.6.4.19. [wait: x-msg-wait request-header](#)
    - 2.6.5. [HTTP return codes](#)

- 2.6.5.1. [HTTP 200: OK](#)
- 2.6.5.2. [HTTP 204: No content](#)
- 2.6.5.3. [MQHTTP0001: No connection factory specified in the Servlet context](#)
- 2.6.5.4. [MQHTTP0001: Could not get connection manager for queueOr Topic using the JNDI name of jndiNameTried](#)
- 2.6.5.5. [MQHTTP40001: Reserved](#)
- 2.6.5.6. [MQHTTP40002: URI is not valid for WebSphere MQ transport for HTTP](#)
- 2.6.5.7. [MQHTTP40003: URI is not valid. @qmqr is only valid on POST](#)
- 2.6.5.8. [MQHTTP40004: Invalid Content-Type specified](#)
- 2.6.5.9. [MQHTTP40005: Bad message header value](#)
- 2.6.5.10. [MQHTTP40006: Header name is not a valid request-header](#)
- 2.6.5.11. [MQHTTP40007: Header name is only valid on ...](#)
- 2.6.5.12. [MQHTTP40008: Header name maximum length is ...](#)
- 2.6.5.13. [MQHTTP40009: Header field header field is not valid for ...](#)
- 2.6.5.14. [MQHTTP40010: Message with Content-Type content type could not be parsed](#)
- 2.6.5.15. [MQHTTP40301: You are forbidden from accessing ...](#)
- 2.6.5.16. [MQHTTP40302: You are forbidden from ...](#)
- 2.6.5.17. [MQHTTP40401: The destination destination name could not be found](#)
- 2.6.5.18. [MQHTTP40501: Method method namenot allowed](#)
- 2.6.5.19. [MQHTTP41301: The message being posted was too large for the destination](#)
- 2.6.5.20. [MQHTTP40401: The destination destination name could not be found](#)
- 2.6.5.21. [MQHTTP41501: The media type character set is unsupported](#)
- 2.6.5.22. [MQHTTP41502: Media-type media-type is not supported ...](#)
- 2.6.5.23. [MQHTTP41503: Media-type media-type is not supported ...](#)
- 2.6.5.24. [MQHTTP41701: The HTTP header Expect is not supported](#)
- 2.6.5.25. [MQHTTP50001: There has been an unexpected problem ...](#)
- 2.6.5.26. [MQHTTP50201: An error has occurred between the HTTP bridge and the queue manager](#)
- 2.6.5.27. [MQHTTP50401: Message retrieval timed out](#)
- 2.6.5.28. [MQHTTP50501: HTTP 1.1 and upwards ...](#)
- 2.6.6. [Supported message types](#)
- 2.6.7. [URI Format](#)

## WebSphere MQ transport for SOAP and WebSphere MQ bridge for HTTP


WebSphere® MQ provides two ways of working with Web services. WebSphere MQ transport for SOAP is an alternative to HTTP for transferring SOAP messages. WebSphere MQ bridge for HTTP enables any HTTP Web client to communicate with WebSphere MQ using the HTTP **GET**, **POST** and **DELETE** verbs.

### [WebSphere MQ transport for SOAP](#)

The WebSphere MQ transport for SOAP provides a JMS transport for SOAP. The WebSphere MQ transport for SOAP is also integrated into other environments such as Microsoft Windows Communication Foundation, WebSphere Application Server, and CICS® Transaction Server.

### [WebSphere MQ bridge for HTTP](#)

With WebSphere MQ bridge for HTTP, client applications can exchange messages with WebSphere MQ without the need to install a WebSphere MQ client. You can call WebSphere MQ from any platform or language with HTTP capabilities.

 This build: January 26, 2011 11:09:34

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20121\_

## 1. WebSphere MQ transport for SOAP

The WebSphere® MQ transport for SOAP provides a JMS transport for SOAP. The WebSphere MQ transport for SOAP is also integrated into other environments such as Microsoft Windows Communication Foundation, WebSphere Application Server, and CICS® Transaction Server.

### [Introduction to WebSphere MQ transport for SOAP](#)

The WebSphere MQ transport for SOAP provides a JMS transport for SOAP. The WebSphere MQ SOAP sender and listener provide a means to call Web services. The listener supports services hosted by .NET Framework 1, .NET Framework 2, and Axis 1.4. The sender supports Web services clients running on .NET Framework 1, .NET Framework 2, Axis 1.4 and Axis2. Clients can be either a WebSphere MQ server or client application. The WebSphere MQ transport for SOAP is also integrated into other environments such as Microsoft Windows Communication Foundation, WebSphere Application Server, and CICS Transaction Server.

### [Installing WebSphere MQ Web transport for SOAP](#)

Use these instructions to install the WebSphere MQ Web transport for SOAP. The installation creates tools to run Web service clients or services using WebSphere MQ as the SOAP transport. The tools are used in the .NET Framework 1, .NET 2, Axis 1.4, or Axis2 SOAP environments.

### [Verifying the WebSphere MQ transport for SOAP](#)

Verify the WebSphere MQ transport for SOAP using the **runivt** command. The command runs a number of demonstration applications and ensures that the environment is correctly set up after installation.

### [Developing Web services for WebSphere MQ transport for SOAP](#)

Use your normal Web service development environment to develop services for use with the WebSphere MQ transport for SOAP.

### [Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP](#)

Use your normal development environment to develop Web service clients for use with the WebSphere MQ transport for SOAP.

### [Deploying Web services using the WebSphere MQ transport for SOAP](#)

Deploy a Web service to one of a number of different server environments and connect to it using WebSphere MQ transport for SOAP.

### [Deploying Web service clients to use WebSphere MQ transport for SOAP](#)

Deploy a Web service client to one of a number of different client environments and connect to a service using WebSphere MQ transport for SOAP.


### [Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server](#)

When you complete this task you will have called a JAX-WS Web service running in WebSphere Application Server from an Axis2 client. The Axis2 client and WebSphere Application Server use the W3C candidate recommendation for the SOAP over JMS protocol running on WebSphere MQ. Use Eclipse Galileo and Rational® Application Developer to build the Web service client and Web service, respectively.

### Reference

WebSphere MQ transport for SOAP reference information arranged alphabetically.

**Parent topic:** [WebSphere MQ transport for SOAP and WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:09:35

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20000\_

## 1.1. Introduction to WebSphere MQ transport for SOAP

The WebSphere® MQ transport for SOAP provides a JMS transport for SOAP. The WebSphere MQ SOAP sender and listener provide a means to call Web services. The listener supports services hosted by .NET Framework 1, .NET Framework 2, and Axis 1.4. The sender supports Web services clients running on .NET Framework 1, .NET Framework 2, Axis 1.4 and Axis2. Clients can be either a WebSphere MQ server or client application. The WebSphere MQ transport for SOAP is also integrated into other environments such as Microsoft Windows Communication Foundation, WebSphere Application Server, and CICS® Transaction Server.

The integration into Microsoft Windows Communication Foundation is part of the WebSphere MQ support for .NET Framework 3.

The WebSphere MQ transport for SOAP is a set of protocols and tools to transport of SOAP messages using JMS over WebSphere MQ. It is packaged in different ways for different application environments as shown in [Table 1](#).

*Table 1. WebSphere MQ transport for SOAP application environments*

	<b>Integrated with additional WebSphere MQ components</b>	<b>Integrated into a framework</b>
<b>Provided as part of WebSphere MQ installation</b>	.NET Framework 1, .NET Framework 2, and Axis 1.4	Windows Communication Foundation (.NET Framework 3) Axis 2 (Client only)
<b>Provided in another software package</b>		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

Integration of the WebSphere MQ transport for SOAP into an application framework simplifies the development and deployment of Web services to WebSphere MQ.

With additional WebSphere MQ SOAP components, you must interact directly with the WebSphere MQ SOAP components to develop and deploy services. Use WebSphere MQ SOAP tools to configure Web services and Web service clients for deployment to WebSphere MQ.

In the integrated environments, development and deployment is simpler. You use the same tools for development and deployment as you would to develop and deploy a SOAP HTTP Web service. You must still configure the WebSphere MQ queues, channels, and queue managers that you require using WebSphere MQ tools.

You can mix and match WebSphere MQ SOAP clients and servers from any of these environments.

### Benefits

The WebSphere MQ transport for SOAP offers existing WebSphere MQ users two principal benefits.

#### Using your WebSphere MQ network to connect existing Web services.

The services might be ones you have written, or services that are provided as interfaces to other packaged software applications you have deployed.

The benefit comes from using your existing WebSphere MQ network to connect Web services. The WebSphere MQ transport has the advantage of being a managed and reliable queued messaging service.

#### Writing new applications, or converting existing applications, to use SOAP rather than WebSphere MQ interfaces.

Typically, applications require a specific WebSphere MQ adapter to be developed to integrate with another application. Adapters have two parts: the connector piece, that puts and gets messages to and from the transport, and the adapter piece that converts data to and from application-specific formats. Integrating each pair of applications is a new challenge.

The benefit of SOAP comes from standardizing on SOAP for defining application interfaces, and then having a choice of transports. You do not need to write application-specific adapters, and you can choose whether to use WebSphere MQ, or HTTP, as the connector. Which transport you choose depends on what qualities of service and connectivity you require.

For existing SOAP over HTTP users, the benefit of WebSphere MQ transport for SOAP comes from using a managed and reliable asynchronous transport. The benefits are twofold:

#### Truly asynchronous programming model for availability and performance.

By using an asynchronous client interface, you can write both short and long running request-reply applications. Long running request-reply applications do not require both the client and service to be available at the same time. A long running request-reply client can be restarted automatically to receive its reply when it is delivered.

#### A ready built managed network that is designed to be reliable and available.

By choosing WebSphere MQ as a transport, you are getting the advantage of using a managed network that provides reliable messaging.

In contrast, transports such as HTTP and FTP over TCP/IP are unmanaged. An unmanaged network is ideal for unpredictable connections: there are fewer management tasks.

## Summary

WebSphere MQ transport for SOAP provides the following components:

- The SOAP/JMS transport binding is used in WSDL documents to bind a SOAP service to a JMS transport. The WebSphere MQ implementation of the SOAP/JMS binding uses a URI that takes either of two forms:

### WebSphere MQ transport for SOAP

```
jms:/queue?&Name=Value&Name=Value...
```

### WebSphere MQ wire format for W3C candidate recommendation

```
jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
```

- The mapping of a SOAP message onto a WebSphere MQ message.
- Two WebSphere MQ SOAP listeners to receive SOAP requests, one for Java and one for .NET Framework 1 or .NET Framework 2. The listeners use .NET or Axis 1.4 to process the SOAP request.
- Two WebSphere MQ SOAP senders to create WebSphere MQ SOAP requests. Web services clients register with a sender to process `jms: SOAP` requests.
- Integration with Windows Communication Foundation (WCF), sometimes known as .NET 3, to send and receive WebSphere MQ Transport for SOAP messages.
- Integration of the client with Axis 2, sometimes known as JAX-WS, to send WebSphere MQ Transport for SOAP or W3C SOAP JMS messages.
- The command, **amqwdeployWMQService** creates development and run time components and scripts to deploy a Web service using the WebSphere MQ transport for SOAP.
- Sample Java and .NET client and service code.
- A script to set the class path, and other utility scripts.

In the integrated environments the sender and listener are integrated into each environment, as are extensions to the development and deployment tools.

### Integration of SOAP and WebSphere MQ

The WebSphere MQ transport for SOAP extends SOAP, and Web services tools and run time, with WebSphere MQ as an alternative transport to HTTP for SOAP. You do not need to modify existing Web services to use WebSphere MQ transport for SOAP as a transport. The transport uses a custom URI format for SOAP/JMS. The W3C URI format for SOAP/JMS is supported in a limited way by Axis2 clients.

### Implementation of WebSphere transport for SOAP on .NET Framework 1, .NET 2 and Axis 1.4

You might want to write your own WebSphere MQ SOAP sender and listener. Use the implementation of WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2, and Axis 1.4 as a guide.

### WebSphere MQ transport for SOAP and Web services reliable messaging

Web services reliable messaging is a protocol for reliably exchanging Web service requests and responses over an unreliable connection. It is best-suited to solve problems of short-lived connection disruption.

**Parent topic:** [WebSphere MQ transport for SOAP](#)

#### Related concepts

[Using WebSphere MQ custom channels for WCF](#)


#### Related information

[Configuring CICS to use the WebSphere MQ transport](#)

[Configuring JMS resources for the synchronous SOAP over JMS endpoint listener](#)

[How to connect to WebSphere MQ](#)

[Windows Communication Foundation](#)

 This build: January 26, 2011 11:10:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts21390\_

## 1.1.1. Integration of SOAP and WebSphere MQ

The WebSphere® MQ transport for SOAP extends SOAP, and Web services tools and run time, with WebSphere MQ as an alternative transport to HTTP for SOAP. You do not need to modify existing Web services to use WebSphere MQ transport for SOAP as a transport. The transport uses a custom URI format for SOAP/JMS. The W3C URI format for SOAP/JMS is supported in a limited way by Axis2 clients.

An additional line of code has to be added to clients in the .NET Framework 1, .NET Framework 2, and Axis 1.4 environments. No additional code is required in Axis 2 and Windows Communication Foundation (WCF) clients. The WebSphere MQ SOAP listener runs services in the .NET Framework 1, .NET Framework 2, and Axis 1.4 environments. The WebSphere MQ transport for SOAP is integrated into some other application server environments, including WCF, CICS, and WebSphere Application Server.

### What is SOAP?

SOAP<sup>1</sup> describes the standardized format of the messages and interaction protocols that applications use to exchange requests, replies, and datagrams. SOAP is independent of the transport used to transfer the messages, and of the application environment that sends and receives the messages. The W3C defines SOAP Version 1.2 succinctly:

*SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment.*<sup>2</sup>

To use SOAP it must be bound to a transport, such as HTTP, e-mail, or WebSphere MQ.

A SOAP protocol binding framework is the set of rules for carrying a SOAP message on top of another protocol, such as HTTP. [SOAP Version 1.2 Part 2: Adjuncts \(Second Edition\)](#) describes the SOAP HTTP binding.

The W3C candidate recommendation, 4 June 2009, [SOAP over Java Message Service 1.0](#), describes the recommendation for the SOAP JMS binding. As JMS is an API specification, and not a transport protocol, the JMS SOAP recommendation does not describe the wire-format of SOAP JMS

messages. It describes the SOAP interaction protocols and the JMS API binding. Consequently, when using the JMS SOAP recommendation you must still use the same JMS implementation for the SOAP client and the SOAP server. It does enable a SOAP JMS application to be run on any implementation of JMS. A JMS implementation can be plugged into a J2EE application server, if both the server and the JMS implementation comply with the JCA specification. WebSphere MQ JMS complies with the JCA specification and can be plugged into a compliant application server.

WebSphere MQ transport for SOAP binding is like the proposed W3C standard, but it is not the same. Its usage is described in the topic [MORFH2 SOAP settings](#). Unlike the W3C candidate recommendation, the SOAP binding is not formally specified. Effectively, it is the HTTP binding, and the service address takes the form, `jms:/queue?name=value&name=value...`, rather than `http://authority/path?query#fragment`. `jms:` is not an officially registered IANA URI scheme.

### What is a Web service?

SOAP enables programs written in different languages, running on different platforms, to communicate using various transport protocols. SOAP is the protocol specification. A Web service is an application that provides a service through a SOAP interface that can be accessed using internet protocols.

An important goal of SOAP is to provide services that clients can use easily. Once you have designed a client to use a service, you can program the call to invoke the service without reference to external documentation. Service interfaces are described in XML, in a WSDL document. The query, `http://authority/path?wsdl`, returns the WSDL description of a SOAP service.

**Tip:** When you deploy a Web service to use WebSphere MQ, also deploy the service to HTTP so that the standard WSDL query works.

### Developing Web services

Web services have a client and a service part. The service is written first, either starting from the interface description in WSDL, or by following the rules for writing the service class. Web service toolkits have utilities to generate WSDL from the interface definition of a class; for example **java2wsdl** or **disco**. They also have tools to generate or class skeletons from WSDL interface descriptions; for example **wsdl2java**, **wsimport**, or **wsdl**. The former is known as bottom up development, and the latter top down.

The **amqwdeployWMQService** command in WebSphere MQ transport for SOAP uses these tools to generate WSDL, client stubs, and client proxies.

Web services are typically written using an integrated development environment targeted at a particular application server environment:

#### Eclipse IDE for Java EE Developers

Creates Web services for Axis 2. Supports JAX-RPC and JAX-WS

#### Rational® Application Developer V7.5

Creates Web services for WebSphere Application Server V7 and previous versions, and also for Axis. Supports JAX-RPC and JAX-WS.

#### WebSphere Integration Developer V6.2

Creates Web services for WebSphere Process Server and WebSphere ESB. Supports JAX-RPC and JAX-WS.

#### Visual Studio 2008 (Version 9)

Creates Web services for .NET Framework 3.5 and earlier (Windows Communication Foundation)

#### Visual Studio 2005 (Version 8)

Creates Web services for .NET Framework 2 and earlier

You can use any of these tools in combination with WebSphere MQ transport for SOAP. Once you have developed a service to use with HTTP, use the **amqwdeployWMQService** tool to deploy the services to use WebSphere MQ as a transport. You can write a new client using the output from the tool, or modify your existing clients to use the WebSphere MQ transport for SOAP.

If WebSphere MQ transport for SOAP is integrated into the application environment, then you do not need use the **amqwdeployWMQService** tool or modify the client code. The client SOAP layer directs client requests that have a URI with the prefix `jms:` to the WebSphere MQ transport for SOAP. The server SOAP layer calls WebSphere MQ transport for SOAP to wait for `jms:` SOAP requests, and returns responses to WebSphere MQ transport for SOAP.

Typically, .NET services have been developed bottom up using Web service annotations in code, and Java services top down, using WSDL interface definitions. The difference in approaches is narrowing, as Java Standard Edition Version 6 supports JAX-WS 2.0, and uses annotations to qualify the definition of service interfaces. It is now as easy to develop Java services bottom up as top down. Which approach you choose is a matter of development method.

The Web services client is written after the service, using the WSDL service definition and generated client stubs and proxies. In some applications, the service definition is not known when the client is written. The client retrieves the service WSDL and creates service requests dynamically. More commonly the service definition is known, but the address to which the service is deployed, is not. The Web service toolkit generates interfaces for the client to use to make service requests. The client provides the service address when it is required. In the third case, the WSDL contains all the information a client needs. The WSDL contains both the interface and address of the service. The code generated by the Web service toolkit has all the information needed by the client to make requests of a service.

You can use any of these three styles with WebSphere MQ transport for SOAP.

### Web service application environments

Web service toolkits require a mapping from the WSDL definition of a service to the streams of bytes that are transferred in SOAP requests and responses. The byte stream is defined by the SOAP specification, and is contained in the SOAP envelope. The SOAP envelope is shown in [Figure 1](#).

Figure 1. SOAP envelope

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
<!-- headers... -->
</soap:Header>
<soap:Body>
<!-- payload or fault message -->
</soap:Body>
</soap:Envelope>
```

Mapping from the SOAP envelope to the language binding and back is part standardized, and part proprietary. The mapping is fundamental to the .NET architecture, and is provided as part of the Common Language Runtime (CLR). The mapping is standardized in Java by JAX specifications. Because the Java mappings are standardized, Java Web service clients and services are portable between different Java-based application environments. JAX-RPC (sometimes called JAX-WS 1.0) is the mapping most in use today. It is supported by Axis 1.4. JAX-WS (sometimes called JAX-WS 2.0) is a greatly improved standard and is likely to replace JAX-RPC rapidly. JAX-WS is supported by Axis 2.0. WebSphere MQ 7.0.1 does not

support JAX-WS and Axis 2.

WebSphere MQ transport for SOAP does not alter the contents of the SOAP envelope, and the contents do not affect the transport. The language bindings do affect the WebSphere MQ transport for SOAP. WebSphere MQ 7.0.1 supports .NET Framework 1, .NET Framework 2, and Axis 1.4 using the code and utilities shipped with WebSphere MQ transport for SOAP. Support for the WebSphere transport for SOAP in .NET Framework 3 and 3.5 is implemented using the WebSphere MQ custom channel for Windows Communication Foundation.

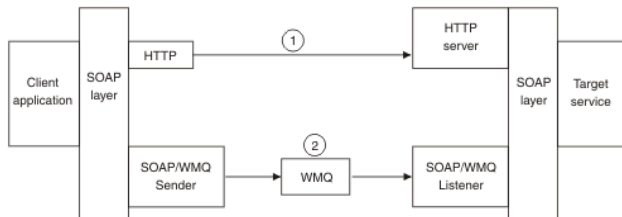
Other SOAP development and run time environments might ship support for WebSphere MQ transport for SOAP, and support different languages. For example, Web services running on CICS® supports languages such as COBOL and PL/1.

**Note:** The mapping used makes no difference to the interoperability of Web services. You can mix and match clients and services written using .NET, JAX-RPC, and JAX-WS mappings.

**What is WebSphere MQ transport for SOAP?**

WebSphere MQ transport for SOAP is a SOAP binding and a Web services toolkit. Together, they enable applications to exchange SOAP messages using WebSphere MQ rather than HTTP. [Figure 2](#) shows WebSphere MQ as an alternative to HTTP as a SOAP transport.

Figure 2. Overview of WebSphere MQ transport for SOAP



SOAP over HTTP is shown as (1) in the diagram. The client SOAP layer converts a request into a SOAP message and the HTTP component sends over TCP/IP. The HTTP server component listens for HTTP requests, typically on the TCP/IP port 80. If the request is for a SOAP service, the HTTP server component calls the SOAP layer to convert the SOAP request into method call. It then returns the response.

SOAP over WebSphere MQ is shown as (2). The client application registers the WebSphere MQ SOAP sender component as a handler for the `javax` protocol with the SOAP layer. The SOAP layer passes SOAP messages addressed to `javax` to the WebSphere MQ SOAP sender. The sender uses the URI in the message to place the message on the request queue with the required qualities of service. The corresponding WebSphere MQ SOAP listener waits for messages on its request queue and calls the SOAP layer to process requests and return responses.

The SOAP sender and listener are normal WebSphere MQ programs. They can be connected to the same queue manager, as in [Figure 3](#), or connected to different queue managers; see [Figure 4](#). The client can be connected by a client connection.

Figure 3. Queues used by SOAP/WebSphere MQ (single queue manager)

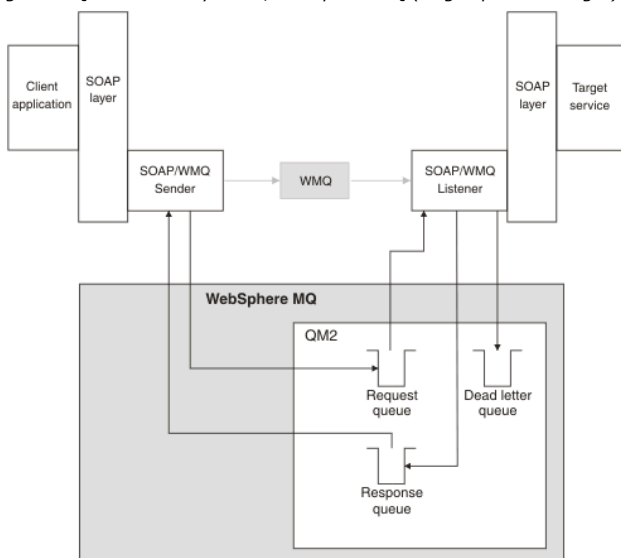
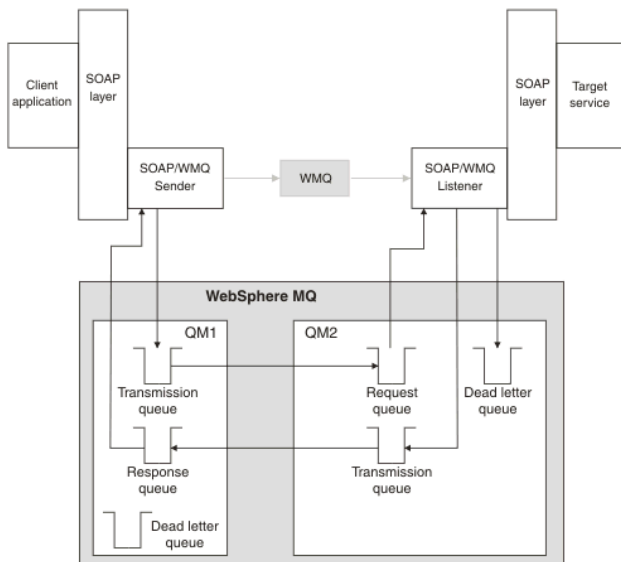


Figure 4. Queues used by SOAP/WebSphere MQ (separate queue managers)



### W3C candidate recommendation for binding SOAP to JMS.

The W3C candidate recommendation defines the SOAP over JMS binding; [SOAP over Java Message Service 1.0](#). Also useful for its examples is [URI Scheme for Java\(tm\) Message Service 1.0](#)<sup>3</sup>.

Some application frameworks, such as WebSphere Application Server v7, have support for the W3C candidate recommendation. Send SOAP requests formatted with a URI compatible with the W3C candidate recommendation using the Axis2 client; see [W3C SOAP over JMS URI for the WebSphere MQ Axis 2 client](#). The Axis2 client sends a SOAP request formatted with either a W3C or a WebSphere MQ transport for SOAP based on URI in the SOAP request.

Axis2 client support for the W3C recommendation is introduced in the 7.0.1.3 fixpack. Support for other clients, and for the SOAP listeners provided by WebSphere MQ is not provided.

**Parent topic:** [Introduction to WebSphere MQ transport for SOAP](#)

#### Related concepts

[Implementation of WebSphere transport for SOAP on .NET Framework 1, .NET 2 and Axis 1.4](#)  
[WebSphere MQ transport for SOAP and Web services reliable messaging](#)

#### Related reference

[SOAP Version 1.2 Part 2: Adjuncts \(Second Edition\)](#)  
[SOAP over Java Message Service 1.0](#)  
[MORFH2 SOAP settings](#)

#### Related information

[Rational Application Developer for WebSphere Software Information Center](#)  
[WebSphere Business Process Management Version 6.2 information center](#)  
[Eclipse Galileo Packages](#)  
[An Overview of Microsoft Visual Studio 2008](#)  
[Creating a Web service for CICS](#)

<sup>1</sup> Historically, the acronym stood for Simple Object Access Protocol.

<sup>2</sup> [W3C: SOAP Version 1.2 Part 0](#)

<sup>3</sup> Look for [URI Scheme for JMS](#), in the W3C specification references, for the latest draft.

This build: January 26, 2011 11:10:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
 ts21392\_

## 1.1.2. Implementation of WebSphere transport for SOAP on .NET Framework 1, .NET 2 and Axis 1.4

You might want to write your own WebSphere® MQ SOAP sender and listener. Use the implementation of WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2, and Axis 1.4 as a guide.

1. A client program uses the appropriate Web services framework in the same way as it would for the HTTP transport. It must also register the `jms:` prefix. The prefix is registered using either the `com.ibm.mq.soap.Register.extension()` Java method or the `IBM.WMQSOAP.Register.Extension()` CLR method.
2. The Axis 1.4 or .NET Framework 1 or 2 framework marshals the call into a SOAP request message exactly as for SOAP/HTTP.
3. A WebSphere MQ service is identified by a URI prefixed with `jms:.` When the framework identifies the `jms:` URI, it calls the WebSphere MQ transport sender code; `com.ibm.mq.soap.transport.jms.WMQSender` (for Axis 1.4) or `IBM.WMQSOAP.MQWebRequest` (for .NET1 and 2). If the framework encounters a URI with an `http:` prefix, it calls the standard SOAP over HTTP sender.
4. The SOAP message is transported by the WebSphere MQ SOAP sender using the request queue. The **SimpleJavaListener** (for Java) or **amqwSOAPNETListener** (for .NET) receives the request message. The WebSphere MQ SOAP listeners are stand-alone processes and are multithreaded with a customizable number of threads.
5. The WebSphere MQ SOAP listener reads the incoming SOAP request, and passes it to the appropriate Web service infrastructure.
6. The Web service infrastructure parses the SOAP request message and invokes the service. The procedure is the same as for a message that

arrived on an HTTP transport.

7. The infrastructure formats the response into a SOAP response message and returns it to the WebSphere MQ SOAP listener.
8. The listener places the message on the response queue, and the message is transferred to the WebSphere MQ SOAP sender. The sender passes it to the client Web service infrastructure.
9. The client infrastructure parses the response SOAP message and hands the result back to the client application.

Each application context is served by a separate WebSphere MQ request queue.

The application context is controlled in Axis 1.4 by ensuring that the WebSphere MQ SOAP listener and service execute in the appropriate directory. Axis 1.4 sets the correct CLASSPATH for the directory.


Application context is controlled in .NET by the WebSphere MQ SOAP listener executing the service in a context created by a call to `ApplicationHost.CreateApplicationHost`. The call specifies the target execution directory. Each service then operates in the directory in which it was deployed.

**amqwdployWMQService** generates the request and response queues. It also generates the infrastructure necessary for handling the queues and deploying services to Axis 1.4.

**Parent topic:** [Introduction to WebSphere MQ transport for SOAP](#)

#### Related concepts

[Integration of SOAP and WebSphere MQ](#)  
[WebSphere MQ transport for SOAP and Web services reliable messaging](#)

 This build: January 26, 2011 11:10:17

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts21394\_

### 1.1.3. WebSphere MQ transport for SOAP and Web services reliable messaging

Web services reliable messaging is a protocol for reliably exchanging Web service requests and responses over an unreliable connection. It is best-suited to solve problems of short-lived connection disruption.

WebSphere® MQ for SOAP takes advantage of using a WebSphere MQ managed and reliable network for passing SOAP messages. Transports such as HTTP and FTP are unmanaged. Unmanaged networks are ideal for unpredictable connections, where the difficulties and costs of managing connections outweigh the benefits of not losing requests and responses.

To overcome the problem of losing files when connections break in unmanaged networks, services like managed FTP build a management layer on top of FTP. The management layer takes over the burden of checking that files have transferred successfully from users, and retransmits missing files if necessary. To use managed FTP, you must have the management software installed at both ends of the connection.

Web services reliable messaging (WSRM) takes a different approach to solving the problem of unreliable connections. Its goal is to transfer Web service requests and responses reliably, without both ends of the connection having to use the same software. Any software, by implementing the Web services reliable messaging protocol, can exchange messages reliably with other software.

When a connection fails, a sender and receiver must preserve the context of the WSRM message transfer, using a generated URI as a key. The sender and receiver keep attempting to establish a new connection. If a new connection is successfully established, the transfer completes. The WSRM specification does not specify how context is preserved, or when a new connection is attempted.

You might decide that only short-lived outages are of interest. For longer outages, you might be prepared to discard transfers that could not be restarted after a time. Similarly, you might be prepared to discard transfers if either the client or service fails. Leaving the user responsible for assuring transfers, places fewer demands on managing the coordination of client and service.

If network outages are long-lived, over 30 minutes or so, or if the client or server fails, there is an increased likelihood that some connections are never re-established. You can no longer rely on WSRM restoring message transfer automatically in an unmanaged way. You have to consider managing the failed WSRM connections, which implies developing software to manage the network of clients and services.

Using WSRM to overcome short outages might significantly reduce dealing with lost messages on a mobile network. If you do not have to assure message delivery, the benefits of reducing message loss might justify the additional cost of developing a WSRM implementation.

SOAP over JMS provides assured message delivery and deals with longer duration outages of the client, the server, and the network. If you are seeking a more reliable quality of service for SOAP than HTTP, which solution do you choose: WebSphere MQ transport for SOAP or WSRM? The answer depends on many factors. Some of the factors to consider are listed:

1. Whether the unreliability is due to connection failure.
2. How long-lived connection failures are.
3. If you can manage both the client and server side of the connection.
4. If the user or an administrator is ultimately responsible for the delivery of messages.


**Parent topic:** [Introduction to WebSphere MQ transport for SOAP](#)

#### Related concepts

[Integration of SOAP and WebSphere MQ](#)  
[Implementation of WebSphere transport for SOAP on .NET Framework 1, .NET 2 and Axis 1.4](#)

#### Related information

[Reliable messaging](#)

 This build: January 26, 2011 11:10:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts21391\_



## 1.2. Installing WebSphere MQ Web transport for SOAP

Use these instructions to install the WebSphere® MQ Web transport for SOAP. The installation creates tools to run Web service clients or services using WebSphere MQ as the SOAP transport. The tools are used in the .NET Framework 1, .NET 2, Axis 1.4, or Axis2 SOAP environments.

### Before you begin

Check the prerequisite products at [WebSphere MQ requirements](#). The installation process does not check for the presence and availability of the prerequisite software. You must verify that the prerequisites are installed.

WebSphere MQ provides a copy of the Axis 1.4 run time. Use this version with WebSphere MQ rather than any other you might have installed. IBM® does not provide technical support for Apache Axis. Contact the Apache Software Foundation if you have technical problems with it.

To run Web services in the .NET Framework 3 SOAP environment, WebSphere MQ uses Windows Communication Foundation. The WebSphere MQ custom channel for Windows Communication Foundation runs Web service clients and services using WebSphere MQ as a transport for SOAP messages.

### About this task

You can install the WebSphere MQ Web transport for SOAP as either a WebSphere MQ Client or Server application. If you have already installed WebSphere MQ as a client or a server on your computer, check that you have installed the components listed.

<mqmtop> represents the high-level directory in which WebSphere MQ is installed.

Carry out the following installation steps.

### Procedure

1. Select the "Java and .Net Messaging and Web services" component for installation.
2. On Solaris and HP-UX, select the "Java Runtime environment" component for installation.
3. Select the Development toolkit for installation.
4. Install and verify WebSphere MQ as described in the Quick Beginning for your platform.
5. Copy the Apache Axis 1.4 run time, `axis.jar` from the `prereqs/axis` directory on the WebSphere MQ installation media. Copy it to the installation directory described in [Table 1](#), [Table 2](#), or [Table 3](#).

#### Windows

```
Copy D:\PreReqs\axis\axis.jar <mqmtop>\java\lib\soap
```

#### AIX®

```
cp -f PreReqs/axis/axis.jar <mqmtop>/java/lib/soap/axis.jar
chown mqm:mqm <mqmtop>/java/lib/soap/axis.jar
chmod 444 <mqmtop>/java/lib/soap/axis.jar
```

#### HP-UX, Solaris, and Linux (all platforms) installation directories

```
cp -f PreReqs/axis/axis.jar <mqmtop>/java/lib/soap/axis.jar
chown mqm:mqm <mqmtop>/java/lib/soap/axis.jar
chmod 444 <mqmtop>/java/lib/soap/axis.jar
```

6. On Windows 2003, run `Aspnet_regiis.exe` to update the script maps to point to the version of the Common Language Run time your are using. Look for the `Aspnet_regiis.exe` utility in `%SystemRoot%\Microsoft.NET\Framework\version-number`.
7. Set the environment variable, `WMQSOAP_HOME`, to point to the WebSphere MQ installation directory.

### Results

Table 1. Windows installation directories

Location	Contents
<mqmtop>\programs\bin	Binary, command, DLL, and executable files
<mqmtop>\programs\java\lib	.jar files
<mqmtop>\programs\java\lib\soap	SOAP .jar files
<mqmtop>\programs\tools\soap\samples	Samples and IVT

Table 2. AIX installation directories

Location	Contents
<mqmtop>/bin	Shell scripts
<mqmtop>/java/lib	.jar files
<mqmtop>/java/lib/soap	axis.jar and other JAX-RPC .jar files
<mqmtop>/tools/samp/soap	Samples and IVT

Table 3. HP-UX, Solaris, and Linux (all platforms) installation directories

Location	Contents
<mqmtop>/bin	Shell scripts
<mqmtop>/java/lib	.jar files
<mqmtop>/java/lib/soap	axis.jar and other JAX-RPC .jar files
<mqmtop>/tools/samp/soap	Samples and IVT

### What to do next

1. For .NET only, you must register the WebSphere MQ transport for SOAP files with the Global Assembly Cache. If .NET is already installed when you install WebSphere MQ, registration is performed automatically at installation. If you install .NET after WebSphere MQ, the registration is performed automatically when the IVT is first run.  
You can run `amqwregisterDotNet` to perform registration. You can also run `amqwregisterDotNet` to force reregistration at any stage. Once made, this registration survives system restarts and subsequent reregistration is not normally necessary.
2. Run the Installation Verification Test.

- If you intend to develop Axis2 client, you must download Axis2 1.4.1 from Apache; see [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#).

**Parent topic:** [WebSphere MQ transport for SOAP](#)

#### Related tasks


[Verifying the WebSphere MQ transport for SOAP](#)

#### Related reference


[amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET Apache software license](#)

#### Related information

[System requirements for WebSphere MQ](#)

 This build: January 26, 2011 11:09:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20360\_

## 1.3. Verifying the WebSphere MQ transport for SOAP

Verify the WebSphere® MQ transport for SOAP using the **runivt** command. The command runs a number of demonstration applications and ensures that the environment is correctly set up after installation.

### Before you begin

The **runivt** command changes the contents of the `soap/samples` directory. To keep the installation image unchanged, copy the samples directory to a temporary location, and run the verification test from the temporary location.

You can run the installation verification as many times as you like.

### About this task

Instead of running the verification test on both .NET and Axis, you might want to run the test only on Axis, or only on .NET.

If you run into problems with the tests and want to start again:

- Stop the queue manager `WMQSOAP.DEMO.QM` using the `immediate` option.
- Stop the listener that has been started in a different window.
- Delete the queue manager.
- Delete the temporary samples directory you created and start again.

On UNIX platforms, you must run the command using an X Window System session.

Carry out the following steps to verify the installation of WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2, and Axis 1.4:

### Procedure

- Copy the `./tools/soap/samples` directory tree to a temporary location.
- Start a command window with the temporary directory as the current directory.
- Use the **runivt** command to start the installation test. You have a number of choices:
  - Run all the tests: `runivt`
  - Run a test on Axis only: `runivt AxisWsd1`
  - Run a test on .NET only: `runivt DotNet`

The list of tests you can run is in the file `ivttests.txt` on Windows and `ivttests_unix.txt` on UNIX platforms.

### Example

**Note:** Some output lines have been split, and only one test is run, to fit the browser window.

*Figure 1. Running the AxisWsd1 test on Windows*

```
C:\IBM\ID\samples>runivt AxisWsd1
Services for demo not yet deployed.
Deploying now. This will take a few seconds, but will not need to be repeated .....
setup environment
clean up old directories and files
The system cannot find the file specified.
Could Not Find C:\IBM\ID\samples\server-config.wsdd
Could Not Find C:\IBM\ID\samples\RegisterDotNET.log
check MQ bits ready
crtmqm WMQSOAP.DEMO.QM completed OK.
strmqm WMQSOAP.DEMO.QM completed OK.
define qlocal(SYSTEM.SOAP.RESPONSE.QUEUE) BOTHRESH(3) completed OK.
define qmodel(SYSTEM.SOAP.MODEL.RESPONSE.QUEUE) BOTHRESH(3) DEFTYPE(PERMDYN)
DEFSOPT(SHARED)
SHARE completed OK.
define qlocal(SYSTEM.SOAP.SIDE.QUEUE) completed OK.
define channel(TESTCHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP) REPLACE completed OK.
define channel(TESTSSLCHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(DES_SHA_EXPORT)
REPLACE completed OK.
deploy the Axis server (generate proxies, etc)
    1 file(s) copied.
deploy the dotNet server (generate proxies, etc)
    1 file(s) copied.
    1 file(s) copied.
Recompile the clients, using the proxies just generated
C:\IBM\MQ\Tools\soap\samples\java\clients\RunIvt.java
```

```


C:\IBM\MQ\Tools\soap\samples\java\clients\SQAxis2Axis.java
C:\IBM\MQ\Tools\soap\samples\java\clients\SQAxis2DotNet.java
C:\IBM\MQ\Tools\soap\samples\java\clients\WsdClient.java
  4 file(s) copied.
Could Not Find C:\IBM\ID\samples\soap\clients\SQAxis2AxisAsync*.java
Note: soap\clients\RunIvt.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: soap\clients\RunIvt.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
  1 file(s) copied.
The system cannot find the file specified.
Configure the client to make sure that Axis clients know about jms:xxx
24-Nov-2009 08:49:18 org.apache.axis.utils.JavaUtils isAttachmentSupported
WARNING: Unable to find required classes (
javax.activation.DataHandler and javax.mail.internet.MimeMultipart).
Attachment support is disabled.
<?xml version="1.0" encoding="UTF-8"?>
<Admin>Done processing</Admin>
C:\IBM\MQ\Tools\soap\samples\ivttests.txt
  1 file(s) copied.
  Demo regenerated
define channel(TESTCHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP) REPLACE completed OK.
define channel(TESTSSLCHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(DES_SHA_EXPORT) REPLACE completed OK.
-----[AxisWsdClient]-----
WMQ transport test: Axis to Axis, using WSDL Axis calls
+++ server: generated\server\startWMQJListener.cmd
--- client: java soap.clients.WsdClient
start WsdClient demo, wsdl port soap.server.StockQuoteAxis_Wmq resolving uri to ...
24-Nov-2009 08:49:26 org.apache.axis.utils.JavaUtils isAttachmentSupported
WARNING: Unable to find required classes (javax.activation.DataHandler and
javax.mail.internet.MimeMultipart). Attachment support is disabled.
'jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&
connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))&
initialContextFactory=com.ibm.mq.jms.NoJndi&
targetService=soap.server.StockQuoteAxis.java&
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE'
Response: 55.25 OK.
-----
C:\IBM\ID\samples>rem generated by DeployWMQService.java at 24-Nov-2009 08:48:51
C:\IBM\ID\samples>call amqwsetcp.cmd
C:\IBM\ID\samples>set CLASSPATH=generated\server [...]
C:\IBM\ID\samples>java com.ibm.mq.soap.util.EndWMQListener -m WMQSOAP.DEMO.QM -q SOAPJ.demos
=====
1 tests run, of which 0 failed.

```

**Parent topic:** [WebSphere MQ transport for SOAP](#)

#### Related reference

[runivt: WebSphere MQ transport for SOAP installation verification test](#)

 This build: January 26, 2011 11:09:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20390\_

## 1.4. Developing Web services for WebSphere MQ transport for SOAP

Use your normal Web service development environment to develop services for use with the WebSphere® MQ transport for SOAP.

### Before you begin

- If you are planning to use the command-line tools supplied with WebSphere MQ transport for SOAP:
  - Create a deployment directory for the service.
  - Start a command window in the directory.
  - For .NET, csc.exe and wsdl.exe must be in the path, and be from the same version of .NET Framework.
  - For Java,
    - Run the **amqwsetcp** command to set up the classpath.
    - An IBM® JRE and a JDK at the same version level must be in the current path. The version level must be at least 1.4.2.
    - Customize the classpath to include the locations of any additional `.jar` libraries, and directories containing `.java` packages, including for the service you are developing. Put the current directory "." in the classpath.
    - Create a directory, relative to the current directory of the command window, corresponding to the package name of the service you are developing.
- Alternatively, use workbench tools that support Web services development. The example development tasks use Microsoft Visual Studio 2008, Eclipse IDE for Java EE Developers and WebSphere Application Server Community Edition.

### About this task

Existing Web services need no modification to work with WebSphere transport for SOAP. The tools supplied with WebSphere MQ transport for SOAP deploy a Web service, and run it using a WebSphere MQ SOAP listener. The tools also generate WSDL, .NET client stubs, and `.java` proxy classes to develop WebSphere MQ transport for SOAP clients.

Follow these steps to create a service, and prepare it for deployment and the generation of clients. Follow the steps in the related tasks to create a service using Eclipse or Microsoft Visual Studio 2008.

### Procedure

1. Develop the service using your normal development environment.
2. Test the service using an HTTP Web services client
3. Follow these steps to prepare the deployment directory:
  - o For Java
    - a. Copy the `.java` file defining the service interface into the deployment directory.
    - b. Copy any `.class` files for the service into the directory corresponding to the package name.
    - c. Check that the classpath can locate all the classes that are required: compile the service `.java` file using **javac**.
  - o For .NET
    - a. Copy the `.asmx` file defining the service into the deployment directory, and
    - b. If you are using the code-behind model, copy any `.dll` files into a `deployment_directory\bin` directory.

#### [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)

Develop an Axis 1.4 Web service to run using WebSphere MQ as your service provider. Use your normal Web service development environment to create a service for deployment to Axis 1.4.

#### [Developing a .NET 1 or 2 service for WebSphere MQ transport for SOAP using Microsoft Visual Studio 2008](#)

Develop the SampleStockQuote Web service for .NET 1 or .NET 2 using Microsoft Visual Studio 2008


#### [Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 2 of connecting an Axis2 Web service client and a Web service deployed to WebSphere Application server using the W3C SOAP over JMS protocol.

**Parent topic:** [WebSphere MQ transport for SOAP](#)

#### Related reference

[amqwdeployWMOService: deploy Web service utility](#)  
[amqwclientconfig: create Axis 1.4 Web services client deployment descriptor for WebSphere MQ transport for SOAP](#)  
[amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET](#)  
[amqswsdl: generate WSDL for .NET Framework 1 or 2 service Transactions](#)  
[URI syntax and parameters for Web service deployment](#)

 This build: January 26, 2011 11:09:44

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20530\_

## 1.4.1. Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse

Develop an Axis 1.4 Web service to run using WebSphere MQ as your service provider. Use your normal Web service development environment to create a service for deployment to Axis 1.4.

### Before you begin

Take into account the requirements for deploying a Web server to the WebSphere® MQ SOAP listener for Axis 1.4.

- The WebSphere MQ SOAP listener for Axis 1.4 requires an IBM® JRE at version 1.4.2 or greater. The JRE, and JDK used for development, must be at the same version level.
- The WebSphere MQ SOAP listener for Axis 1.4 requires the `axis.jar` installed with WebSphere MQ. Change the build path in your development environment to reference the `axis.jar` file installed with WebSphere MQ, rather than the `axis.jar` files installed with the development environment.
- Up to, and including, WebSphere MQ V7.0.1, the WSDL generated for the deployed service is RPC/encoded. From V7.1, you can also request RPC/literal style WSDL. The generated WSDL is used only for deployment. You can define the service using WS-I compliant WSDL.

### About this task

Create the service using your normal Web services development environment.

In this task, we use the freely available open source Eclipse Java EE IDE for Web Developers, known as Galileo. For the application server, we use WebSphere Application Server Community Edition v2.1 (Community Edition), based on Geronimo. See the related tasks for information about how to obtain, install, and configure the IDE and server.

Test the service using HTTP as a transport using the Web Services Explorer provided in the IDE before verifying the service by deploying to the WebSphere MQ transport for SOAP. Alternatively, generate an HTTP client proxy and test the service using your own client code.

You might follow these steps to develop a bottom-up Web service. Use the sample program `StockQuoteAxis.java` as an example.

### Procedure

1. Start Eclipse IDE for Java EE Developers with a new workspace.
2. Configure the workspace to use Java50, WebSphere Application Server Community Edition 2.1.4 does not work with Java60.
  - a. **Window > Preferences > Java > Installed JREs > Add... > Standard VM > Next > Directory ...**
  - b. Browse to the install directory of **Java50 > OK > Finish**
  - c. Check the **Java50** JRE > OK
3. Add the Community Edition runtime environment and start Community Edition.
  - a. **Window > Preferences > Server > Runtime Environments > Add...**
  - b. Select **IBM WASCE v2.1** from the **New Server Runtime Environment** list > Check **Create a new local server > Next** If **IBM**

**WASCE 2.1** is not in the list you need to complete two other tasks:

- i. Install WebSphere Application Server Community Edition.
- ii. Install the Eclipse update for Community Edition.

Find the details at [WebSphere Application Server Community Edition](#)

- c. Browse to the Application Server Installation Directory > **OK** > **Finish** > **OK**.
  - d. Right-click **IBM WASCE v2.1 server** in the servers view > **Start**  
**Tip:** You can manage WASCE in Eclipse: Right-click **IBM WASCE v2.1 server** > **Launch WASCE Console**. the default **Username** and **Password** are `system` and `manager`.
4. Set up the server and runtime for Web services.
    - a. **Window** > **Preferences** > **Web services** > **Server and Runtime**
    - b. Select **IBM WASCE v2.1 Server** as the server.
    - c. Leave **Apache Axis** as the Web service runtime.
  5. Create a Dynamic Web Project.
    - a. **File** > **New** > **Dynamic Web Project**. Name the project `StockQuoteAxis`.
    - b. Check **Add project to an EAR** > **New...**
    - c. In the EAR Application Project page type the **Project name** `StockQuoteAxisEAR` > **Finish** Reply **OK** in response to the dialog box suggesting you switch to the Java EE perspective, or stay Java perspective to follow these instructions exactly.
    - d. **IBM WASCE 2.1 server** is selected as the target runtime. Accept it, and the other defaults > **Finish**. Reply **OK** in response to the dialog box suggesting you switch to the Java EE perspective, or stay Java perspective to follow these instructions exactly.
  6. Import the `StockQuoteAxis.java` sample program
    - a. Open the **StockQuoteAxis** Web project > Right click the `src` folder > **Import...**
    - b. Select **General** > **File System** > **Next**
    - c. Browse to `WMQ Install directory\tools\soap\samples\java\server` > check `StockQuoteAxis.java` > **Finish** You must highlight the server directory in the left window to see the files it contains, in the right window.
  7. Correct the compile error by moving `StockQuoteAxis.java` into its correct package.
    - a. Open `StockQuoteAxis.java` and right click the problem > **Quick Fix**
    - b. Double click **Move 'StockQuoteAxis.java' to package 'soap.server'** > **Save**.
  8. Create a Web service from `StockQuoteAxis.java`.
    - a. Right-click `StockQuoteAxis.java` > **Web Services** > **Create Web service** > **Next**. Accept the default configuration for the service:
 

**Web service type**  
Bottom up Java beanWeb Service

**Service implementation**  
`soap.server.StockQuoteAxis`

**Server**  
IBM WASCE v2.1 server

**Web service runtime**  
Apache Axis

**Service project**  
`StockQuoteAxis`

**Service EAR project**  
`StockQuoteAxisEAR`

**Configuration**  
No client generation
  9. Select the methods to access and the style of Web service > **Next**. Start the server if prompted.
    - a. Leave all the methods selected.
    - b. Select document/literal (wrapped) style.
  10. **Finish**  
When the service has deployed, look in the `WebContent\wsdl` folder in the `StockQuoteAxis` Web project and find the generated `StockQuoteAxis.wsdl` file.
  11. Test the service using HTTP with the Web Services Explorer.
    - a. Right-click `StockQuoteAxis.wsdl` > **Test with Web Services Explorer**.
    - b. Click the operation `getQuote` in the `StockQuoteAxisSoapBinding` actions in the **Web Services Explorer** window.
    - c. Type `ibm` in the **symbol** input field > **Go**
  12. Test the service using WebSphere MQ transport for SOAP.  
To deploy the service, use **SimpleJavaListener**, which is in `com.ibm.mq.soap.jar`. You have to add the WebSphere MQ Java and SOAP libraries to the build path.
    - a. Right click the **StockQuoteAxis** Web project > **Build Path** > **Configure Build Path...**
    - b. Click the **Libraries** tab > **Add External Jars...** Browse to `WMQ Install directory\java\lib`. and select all the `.jar` files > **Open** > **Add External Jars...** Browse to `WMQ Install directory\java\lib\soap` and select all the `.jar` files > **Open** > **OK**.
    - c. In the Project Explorer, Right click `StockQuoteAxis\Referenced Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class` \ `SimpleJavaListener` > **Run as...> Run Configurations...** Select **Java Application** in the navigator.  
**Tip:**  
 If there is no configuration for `SimpleJavaListener`, click the **New launch configuration** icon on the Create, manage, and run configurations page of the **Run configurations** wizard,  
 The **SimpleJavaListener** has no command to stop it. To monitor or stop **SimpleJavaListener**, open the **Debug perspective** in Eclipse.

- d. Open the **(x)= Arguments** tab. In the **Program arguments** input area type the parameters to **SimpleJavaListener**. For this example, type

```
-u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=(  
&initialContextFactory=com.ibm.mq.jms.NoJndi  
&targetService=StockQuoteAxis  
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
```

**Note:** The target service is `StockQuoteAxis` to match the target service name created in the service deployment descriptor, `StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd`. `amqwdeployWMQService` creates a target service called `soap.server.StockQuoteAxis`. In this example, you are using the same `StockQuoteAxis.class` and `service-config.wsdd` as the HTTP server.

- e. On the same tab, configure **Working directory** to reference the `server-config.wsdd` file:  
`${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}`

- a. Create the WebSphere MQ resources, if necessary.  
 The following resources required:

**Queue manager**

QM1

**Request queue**

REQUESTAXIS

**Response queue**

SYSTEM.SOAP.RESPONSE.QUEUE

- b. **Run**

Errors are written to the console. If the console remains blank, **SimpleJavaListener** has started ok.

- c. To test the deployment, run a `StockQuoteAxis` client, developed in the task, [Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse](#).

### Example: StockQuoteAxis sample program

The sample Java Web service, `StockQuoteAxis.java`, is installed in `WMQ install directory\tools\soap\samples\java\server`. `StockQuoteAxis.java`, [Figure 1](#), has four methods:

1. `float getQuote(String symbol)`
2. `void getQuoteOneWay(String symbol)`
3. `int asyncQuote(int delay)`
4. `float getQuoteTran(String symbol)`

Figure 1. `StockQuoteAxis`

```
package soap.server;
import java.lang.Thread;
import java.io.PrintWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }
    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,  
exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

#### What to do next

Deploy the service from the command line using the WebSphere MQ Transport for SOAP tools instead of Eclipse. Use the command `amqwdeployWMQService`.

The command has an option, `axisDeploy`, which deploys the service by creating an Apache Axis 1.4 deployment descriptor. The WebSphere MQ SOAP listener runs the service. The SOAP listener is called `SimpleJavaListener` and is supplied with the WebSphere MQ transport for SOAP.

**Parent topic:** [Developing Web services for WebSphere MQ transport for SOAP](#)

#### Related tasks

[Developing a .NET 1 or 2 service for WebSphere MQ transport for SOAP using Microsoft Visual Studio 2008](#)

[Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)

#### Related information

[IBM Redbooks: Experience Java EE! Using WebSphere Application Server Community Edition 2.1 WebSphere Application Server Community Edition Eclipse Galileo Packages Web services - Axis](#)

This build: January 26, 2011 11:09:45

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
developingjax-rpcjavaservice

## 1.4.2. Developing a .NET 1 or 2 service for WebSphere MQ transport for SOAP using Microsoft Visual Studio 2008

Develop the SampleStockQuote Web service for .NET 1 or .NET 2 using Microsoft Visual Studio 2008

### About this task

Create the StockQuote service with a code-behind implementation using Visual Studio 2008.

### Procedure

- Create a template for the service, and check that it runs on HTTP.
  - Start Visual Studio 2008 > **File** > **New** > **Project....** Select **C#** Project Type, **.NET Framework 2**, and **ASP.NET Web Service Application**. Type the **Name:** and **Solution Name:** StockQuoteDotNet > **OK**
  - Right click **Service1.asmx** in the Solution Explorer > **Rename** > StockQuote.asmx.
  - Change the code fragment `public class Service1` to `public class StockQuote`.
  - Right click **StockQuote.asmx** in the Solution Explorer > **Open with... > XML Editor**. Change `Class="StockQuoteDotNet.Service1"` to `Class="StockQuoteDotNet.StockQuote"`
  - Change the code fragment `[WebService(Namespace = "http://tempuri.org/")]` to `[WebService(Namespace = "http://stock.samples/")]`.
  - Remove the line of code `[ToolboxItem(false)]`.
  - Check everything is correct so far: **Debug** > **Start Debugging (F5)**. Verify the output in Explorer.
- Add the methods from the sample `SQDNNonInline.asmx.cs`, and test the service on HTTP.
  - Open `WMQ install directory\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs` and replace the `HelloWorld` method with the four `Quote` methods; see [Figure 1](#).
  - Build** > **Rebuild** the solution > Right click one of the **Thread** in error > **Resolve** > Using **System.Threading**.
  - Press F5 to start debugging. The service is not conformant to WS-I Basic Profile v1.1. You have the choice of either changing the `WebMethod` annotation from `[SoapRpcMethod]` to `[SoapDocumentMethod]`, or removing the annotation `[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]`.
  - Press F5 to verify the implementation using HTTP.
- Generate WSDL, clients, and run the service using WebSphere® MQ transport for SOAP.
  - Open a command window in the project directory tree, where the `StockQuote.asmx` is stored.
  - (Optional) Use `amqswdeployWMQService` to generate artifacts. The queue manager must be started:
 

```
amqswdeployWMQService -f StockQuote.asmx
-u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```

 All artifacts are created in the `./generated` directory tree.
  - (Optional) Generate just the WSDL for calling the service using WebSphere MQ transport for SOAP.
 

```
amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
StockQuote.asmx StockQuote.wsdl
```
  - Run the .NET listener. Either use `.\generated\server\startWMQNListener.cmd` or type the command:
 

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```
- Test the service using a client generated from the WSDL, or using the clients generated by `amqswdeployWMQService`.

### Sample code

The sample .NET Web service, `StockQuoteDotNet`, is installed in `WMQ install directory\tools\soap\samples\dotnet`. The Web service binding of the published samples differ slightly from the binding used in the task. The task uses the defaults used in Visual Studio 2008.

There are two examples of .NET Framework 1 and .NET Framework 2 Web services. `StockQuoteDotNet.asmx`, is an inline service. `SQDNNoninline.asmx`, is a code-behind Web service implemented by `SQDNNoninline.asmx.cs`.

`StockQuoteDotNet` has four methods:

- `float getQuote(String symbol)`
- `void getQuoteOneWay(String symbol).`
- `int asyncQuote(int delay)`
- `float getQuoteDOC(String symbol)`

Figure 1. Inline service: StockQuoteDotNet.asmx

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
    [WebMethod] [ SoapRpcMethod(OneWay=true) ]
    public void getQuoteOneWay(String symbol) {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getQuoteOneWay was invoked.");
    }
    [WebMethod] [SoapRpcMethod]
    public float getQuote(String symbol) {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }
    [WebMethod] [SoapRpcMethod]
    public int asyncQuote(int delay) {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getQuoteDOC(String symbol) {
        return 77.77F;
    }
}
```

Figure 2. Code-behind: Design SQDNNonInline.asmx

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Figure 3. Code-behind: Implementation: SQDNNonInline.asmx.cs

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{
    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }


    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

Parent topic: [Developing Web services for WebSphere MQ transport for SOAP](#)

#### Related tasks

[Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)  
[Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)

 This build: January 26, 2011 11:09:50

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20536\_

### 1.4.3. Developing a JAX-WS EJB Web service for W3C SOAP over JMS

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 2 of connecting an Axis2 Web service client and a Web service deployed to WebSphere® Application server using the W3C SOAP over JMS protocol.

#### Before you begin

Use Rational® Application Developer to create the EJB Web service. The Web service wizard in Rational Application Developer has an option to



create a Web service using the W3C candidate recommendation for the SOAP over JMS binding. Rational Application Developer 7.54 is required. The exercise used Rational Application Developer included in Rational Software Architect for WebSphere Software v7.5.5.1,

The EJB is deployed to WebSphere Application Server from Rational Application Developer as part of this task. You must complete [Configuring WebSphere Application Server to use W3C SOAP over JMS](#)

To create the WSDL actually used in the task, you must first complete the task, [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#). Then you can either import the WSDL from the Dynamic Web project in the Eclipse Galileo workspace, or from the running HTTP Web service deployed to WASCE.

WebSphere Application Server might still be running as a result of doing [Configure WebSphere Application Server resources](#). If it is not, you can start it from the Servers view in RAD.

### About this task

In this task, you redeploy the `StockQuoteAxis` service from running as a JAX-RPC Axis service run by the `SimpleJavaListener` using WebSphere MQ transport for SOAP, to being a JAX-WS service running in WebSphere Application server using the W3C SOAP over JMS protocol.

There are two parts to migrating the service from the `SimpleJavaListener` to WebSphere Application Server:

1. Generate the Web service interface from the WSDL for the service using the Top-down EJB Web service wizard in Rational Application Developer.
2. Implementing the service by importing the WebSphere MQ SOAP sample `StockQuoteAxis.java`.

An alternative approach would have been to generate the service bottom up, from the `StockQuoteAxis.java`. However, to be sure that the interface to the migrated service is identical, the top-down approach is better, as it uses the same WSDL.

The Web service is developed for the EJB container and not the Web container because the JMS support is part of the EJB container.

### Procedure

1. Start Rational Application Developer, and verify WebSphere Application Server is running.
  - a. Start Rational Application Developer in a new workspace.
  - b. Open the Java EE perspective.
  - c. Open the Servers tab at the bottom of the display, and check WebSphere Application Server is running.
    - If there is no WebSphere Application Server v7.0 in the view, right-click in the view > **New** > **Server**. Follow the choices in the wizard to create a WebSphere Application Server v7.0 instance.
    - If the server is present, but not started, click the green arrow head on the right of the view to start it.
    - To verify the properties and get quick access to the server logs, right-click **WebSphere Application Server v7.0 at localhost** > **Properties** > **WebSphere Application Server**.
    - To administer the server, either use an external browser and open the URL, `http://localhost:9061/ibm/console/unsecureLogon.jsp`, or right-click **WebSphere Application Server v7.0 at localhost** > **Run administrative console**.
    - The default setting is to publish automatically. Many people prefer deploy updates to the server manually. Double-click **WebSphere Application Server v7.0 at localhost**, and expand the Publishing twisty in the upper right corner of the Overview window. Click **Never publish automatically**.
    - Another default to alter is to clear the **Terminate server on workbench shutdown** checkbox in the lower left corner of the Overview window.
2. Create the JEE projects You must create an Enterprise Application Project (EAR) and an Enterprise Java Bean (EJB) Project.
  - a. **File** > **New** > **Enterprise Application Project**. Name the project `W3CJMSEAR` > **Finish**.  
The defaults must identify WebSphere Application Server v7.0 as the target runtime, and EAR version 5.0. The default configuration must be selected.
  - b. **File** > **New** > **EJB Project**. Name the project `W3CJMSEJB`. Select `W3CEARJMS` as the **EAR Project Name** > **Next**.  
The default EJB module version is 3.0 and the default configuration is used again.
  - c. Clear the **Create an EJB Client JAR module** checkbox > **Finish**.

3. Generate and deploy the EJB Web service from the `StockQuoteAxis` WSDL.
  - a. **Run** > **Launch the Web Services Explorer**.
  - b. Select the WSDL page using the icons in the upper right corner of the Web Services Explorer window > click **WSDL main** in the Navigator.
  - c. In the Actions window, type in or browse to the WSDL URL for `StockQuoteAxis.wsdl`.  
If you have WASCE running with `StockQuoteAxis` deployed as an HTTP service, the URL is:  
`http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`  
If you have the WSDL in the file system, the URL might be:  
`File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl`
  - d. Click the line containing the imported URL in the Navigator tree.  
It is the line immediately following **WSDL Main**, if it is the first WSDL you have imported into Web Services Explorer.
  - e. In the Actions window, click **Launch Web Service Wizard** > **Web Service Skeleton** > **Go**
  - f. In the Web Service wizard, select **Top down EJB Web Service**  
Select or verify the Configuration using information from [Table 1](#) Check **Overwrite files without warning** > **Next**.

Table 1. Top down EJB Web service configuration

Field	Value
Server	WebSphere Application Server v7.0
Web service run time	IBM WebSphere JAX-WS
Service project	W3CJMSEJB
Service EAR project	W3CJMSEAR
Configuration:	No client generation

- g. On the page subtitled, Specify options for creating a WebSphere JAX-WS EJB Top Down Web Service, check the **Switch to JMS binding** box. Also check **Enable Wrapper Style**, **Copy WSDL to project**, and **Generate Web Service Deployment Descriptor** > **Next**.

- h. On the page titled, WebSphere JAX-WS JMS Binding Configuration, check **Use SOAP/JMS interoperability protocol** and provide values from [Table 2](#), leaving other fields blank > **Next**.

Table 2. WebSphere JAX-WS JMS Binding Configuration

Field	Value
JMS destination	queue
Destination JNDI name:	requestaxis
JMS connection factory	qml
Reply to Name	W3CJMSEAR
Configuration:	replyaxis

- a. On the page titled, WebSphere JAX-WS Router Project Configuration, type `qmlas` in the **ActivationSpec JNDI name** field > **Next**. RAD takes about 30 seconds to a minute to generate and deploy the project.
- b. Ignore the options in the Web Service Publication page > **Finish**.
4. Check the generated WSDL.  
You asked for the service-specific WSDL to be generated and saved in the project.
- a. In the Enterprise Explorer navigator, open the folder **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl**. Double click `StockQuoteAxis.wsdl` to open it in the WSDL editor.  
Inspect the binding; you see the JMS url:  

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qml&targetService=StockQuoteAxis
```
5. Optional step: Bind the EJB to SOAP over HTTP using JAX-WS.  
Providing two bindings to the EJB gives clients the choice of SOAP bindings to call the Web service. It also provides clients with the means to query the Web server to obtain its WSDL, using HTTP.  
The steps to bind an EJB to SOAP over HTTP are not included as part of the task.
6. Implement and redeploy `StockQuoteAxis` using the sample `StockQuoteAxis.java`
- a. In the Enterprise Explorer navigator, open the folder **W3CJMSEJB** > **Services** Double click `StockQuoteAxisService` to open the implementation class in a Java editor.
- b. Open the `StockQuoteAxis.java` sample program in the *WebSphere MQ Installation directory\tools\soap\samples\java\server* folder > Select all the methods, but not the class name > **Copy**.
- c. In `StockQuoteAxisSoapBindingImpl.java` select all the methods, but not the class name, and paste in the methods from `StockQuoteAxis.java`.
- d. Add a print statement to output to the WebSphere Application Server console when the service is called. Change the `getQuote(String symbol)` method:  

```
public float getQuote(String symbol) {
    System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
        + symbol);
    return ((float) 55.25);
}
```
- e. Correct the imports: **Source** > **Organize imports** > **Save**.
- f. Correct the three errors due to the implementation not matching the interface.  
The errors are due to three of the methods in `StockQuoteAxis.java` throwing exceptions, and the WSDL for the service not containing any fault messages. The problem is diagnosed as being a mismatch between the method signatures and the method Web service annotations.  
Either annotate the methods with `@WebFault` and regenerate the WSDL, or keep the interface unchanged, and remove the exceptions. To keep the interface the same, remove the three `throws exception` from the method signatures > **Save**.


### What to do next

[Deploying to an Axis2 client using W3C SOAP over JMS](#)

Parent topic: [Developing Web services for WebSphere MQ transport for SOAP](#)

### Related tasks

[Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)  
[Developing a .NET 1 or 2 service for WebSphere MQ transport for SOAP using Microsoft Visual Studio 2008](#)  
[Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server](#)  
[Configuring WebSphere Application Server to use W3C SOAP over JMS](#)  
[Deploying to an Axis2 client using W3C SOAP over JMS](#)

 This build: January 26, 2011 11:10:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22551\_

## 1.5. Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP

Use your normal development environment to develop Web service clients for use with the WebSphere MQ transport for SOAP.

### Before you begin

1. Create the service. You can use one of the examples in [Developing Web services for WebSphere MQ transport for SOAP](#).
2. Make choices about how you are going to develop, deploy and use the clients, and where to go to get the WSDL for client generation.
  1. **Decide on your approach to developing clients and services for WebSphere MQ transport for SOAP.**

There are two approaches.

- a. Use standard development tools, develop an HTTP service and client, and then use the URL for WebSphere MQ transport for SOAP.
- b. Use the tools and samples supplied with WebSphere MQ transport for SOAP.

If you take the HTTP route, you can run the service on an HTTP server and also run it using WebSphere MQ transport for SOAP. To run it using WebSphere MQ transport for SOAP, configure the appropriate WebSphere MQ listener for SOAP and set up the paths and

deployment descriptors to run the service. The tools provided by WebSphere MQ transport for SOAP do the configuration for you. Alternatively, you can configure the environment to run the listeners.

The tools supplied with WebSphere MQ transport for SOAP are useful in getting started and learning how to deploy the transport. For production work, there are benefits in using standard tools, and deploying the same service accessible to different SOAP transports.

## 2. Decide on the type of client to develop

You must decide what type of Web service client to develop. The choice depends on whether you know the service interface and the address of the service.

If the interface is known, use Axis or .NET tools to generate proxy client classes from the service interface. The proxy client classes make it easier to write a client to call the service. If the location of the service is known when you develop the client, then use the static proxy interface. If the location of the service changes, for example if the service is redeployed onto a production server, then use the dynamic proxy interface.

If the service interface is not known at the time you develop a client, on Axis, you can create a Dynamic Invocation Interface (DII) client for Axis 1.4. A DII client uses a generic interface to call any service. To pass parameters to a particular service correctly, you need to build the specific service interface programmatically. Build the interface programmatically in the client, or by loading the WSDL for the service into the client. On Axis2, you can create a Dispatch client. The Dispatch client uses a document model to describe the client request, whereas a DII client uses a call model. Both work on building the request dynamically.

## 3. Obtain the WSDL for the service

Except for the case of the service interface being built programmatically, you must first obtain the service WSDL in order to create a Web service client. The service WSDL is obtainable from three different sources:

- Directly from the Web service implementation using a tool such as **java2wsdl** (Axis) or **disco** (.NET).
- By querying the Web service using the URL: `Web service http url?wsdl`.
- From a file, either on a file system, or from a registry such as UDDI or WebSphere Service Registry and Repository.

**Note:** If the service is not accessible using HTTP, then the WSDL query does not work. The service itself might only be available using the WebSphere MQ transport for SOAP.

The WSDL generated by **amqwdeployWMQService** is not the same as WSDL generated using **java2wsdl** or **disco**. The generated WSDL is also different to any WSDL you might have started with to create the service "Top Down". On Axis, the `server-config.wsdd` deployment descriptor maps the SOAP message produced by a client to an operation and service. **amqwdeployWMQService** generates a different deployment descriptor from Eclipse.

The WSDL you use to build clients depends on how the service is deployed:

### Deployed using amqwdeployWMQService

Use the WSDL generated by **amqwdeployWMQService**. Specify the `-w` flag and select `rpcLiteral` WSDL. For compatibility, you can select `rpcEncoded` WSDL. `rpcEncoded` WSDL works only with .NET and Axis 1.4 clients.

### Manual deployment using SimpleJavaListener

Use one of the following WSDL files:

- WSDL used to define the service, or stored in a repository.
- WSDL generated from the service by **java2wsdl**.
- WSDL queried using the URL `Web service http url?wsdl`, if available from an HTTP server. You might run a tool such as Web Services Explorer to import the service definition directly into Eclipse.

You might need to change the URI for the service. Change it from the address of the HTTP service, to the URI for the WebSphere MQ transport for SOAP.

### Manual deployment using amqSOAPNETListener.

Use one of the following WSDL files:

- WSDL used to define the service, or stored in a repository.
- WSDL obtained from the .NET service class (.asmx). using **disco**.
- WSDL queried using the URL `Web service http url?wsdl`, if available. You might run a tool such as Web Services Explorer to import the service definition directly into Eclipse.
- WSDL obtained by running **amqswsdl** against the .NET service class (.asmx).

You might need to change the URI for the service. Change it from the address of the HTTP service, to the URI for the WebSphere MQ transport for SOAP.

### Deployed to Windows Communication Foundation

Obtain the service WSDL by using the URL `Web service http url?wsdl`. The service must be defined with the `serviceMetadata` behavior configuration as part of the service definition.

### Deployment to a different server platform.

Follow the guidance provided with the platform about how to obtain the correct service WSDL.

## About this task

Develop clients using standard development tools. The following tasks illustrate how to build clients for .NET 1 and 2, Axis 1.4 (JAX-RPC) and Axis2 (JAX-WS). For Windows Communication Foundation, see the related tasks links.

### [Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse](#)

Develop an Axis 1.4 Web service client to run using the WebSphere MQ transport for SOAP.

### [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)

Develop an Axis2 Web service client to run using the WebSphere MQ transport for SOAP. The sample Axis2 clients provided with WebSphere MQ transport for SOAP are listed, and the **wsimport** command used to generate proxies.

### [Developing a .NET 1 or 2 client for WebSphere transport for SOAP using Microsoft Visual Studio 2008](#)

Develop an .NET 1 or 2 Web service client to run using the WebSphere MQ transport for SOAP.

**Parent topic:** [WebSphere MQ transport for SOAP](#)

## Related concepts

[Simple request-reply client and server WCF sample](#)

 This build: January 26, 2011 11:09:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20432\_

## 1.5.1. Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse

Develop an Axis 1.4 Web service client to run using the WebSphere® MQ transport for SOAP.

### Before you begin

You must have the service available. If you are following the task as a practical exercise, use the workspace and service you created in the task, [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#). You must have an application server running in eclipse that supports Axis 1.4 Web services. In this task, we use the freely available WebSphere Application Server Community Edition Version 2.1.4. It is configured as part of the task [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#). You might also use Tomcat 6, which is a smaller open source application server.

### About this task

The task shows the development of three types of client for the sample StockQuoteAxis service using Eclipse running on Windows. The clients are a static and dynamic client developed using the client proxy, and a DII client.

Two alternative approaches to generating the client proxies from WSDL are illustrated:

1. Generating client proxies using **amqwdeployWMQService**.
2. Importing WSDL into Eclipse, and using the Web service wizard to generate the client proxies.

### Procedure

1. Start the Eclipse IDE for Java EE developers.
2. Create a Java project called `StockQuoteAxisClient`:
  - a. Switch to the Java perspective > **File** > **New** > **Java Project**. In the **Project name** field of the Create a Java Project page type, `StockQuoteAxisEclipseClient`. Make sure that the execution environment is either **J2SE-1.4** or **J2SE-1.5** > **Next**.
  - b. On the Java Settings page, select the **Libraries** tab > **Add External JARs...**
  - c. Browse to `WMQ program directory/java/lib` and select all the `.jar` files > **Open**.
  - d. Browse to `WMQ program directory/java/lib/soap` and select all the `.jar` files > **Open**. You must have installed `axis.jar` from the WebSphere MQ installation media into this directory.
  - e. The **Library** tab now references all the `.jar` files needed to build the client. Click **Finish**.
3. Follow one of these two approaches to create proxies in Eclipse for the sample StockQuoteAxis Web service:
  - o Generate the client proxies using **amqwdeployWMQService**.
    - a. Create a queue manager. For the task create `QM1` as the default queue manager.
    - b. Create a working directory, `samples`. Copy the `StockQuoteAxis.java` sample program into `samples/soap/server`.
    - c. Modify `amqwsetcp.cmd` in `WMQ install directory/bin` to include the current directory in the class path.
    - d. Open a command window in `samples` and run the modified **amqwsetcp** command
    - e. Create WSDL for the StockQuoteAxis service by running the command,
 

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWSDL
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.NoJndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

**Remember:** Use `"/`, rather than `."` or `"\"` when using Java commands.

**Tip:** Rather than import the generated proxies into Eclipse, you can import the generated WSDL from `.samples/generated`. The resulting proxies differ in two ways:

      - i. The package names are different - which you can refactor.
      - ii. The Eclipse generated proxies include an additional helper class, `StockQuoteAxisProxy.java`
    - f. Create the client proxies for the StockQuoteAxis service by running the command:
 

```
amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
-u "jms:/queue?destination=REQUESTAXIS
&initialContextFactory=com.ibm.mq.jms.NoJndi
&connectionFactory=(connectQueueManager(QM1)binding(auto))"
```
    - g. Import the client proxies into `StockQuoteAxisClient`:
      - i. Right click **StockQuoteAxisClient\src** > Select **File System** > **Next** > **Browse...** > find the folder `.\samples\generated\client\remote\soap\server` > **OK**.
      - ii. Select **server** in the right pane of the Import page > **Finish**.
    - h. Refactor the package name to `soap.server`.
      - i. Right click the package containing the client proxies > **Refactor** > **Rename**. Type the **New name:** `soap.server` > leave the selected defaults for the other choices > **OK**. All the errors are fixed.
  - o Generate the client proxies using Eclipse.
 

You have a choice of ways of obtaining the WSDL for the service. In this example, the service has been deployed to WebSphere Application Server Community Edition and you obtain the WSDL from the Web server. The deployment is described in the task [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#),

    - a. In Eclipse, switch to the Web perspective, and check that the WebSphere Application Server Community Edition v2.1 Server is running and StockQuoteAxis is deployed and synchronized.
    - b. Import the WSDL into Web Services Explorer:
      - i. Click the **Web Services Explorer** icon in the action-bar, or click **Run** > **Launch the Web Services Explorer**.
      - ii. Click the WSDL page icon in the upper right corner of the Web Services Explorer to switch to the WSDL page.
      - iii. Click **WSDL Main** in the Navigator window of the Web Services Explorer.
      - iv. Type in the URL of the Web service, followed by `?WSDL`. The URL for StockQuoteAxis, deployed in the task [Developing a JAX-](#)

[RPC service for WebSphere MQ transport for SOAP using Eclipse](http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl), is:

```
http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
```

- c. Generate the client proxies:
  - i. In the Web Services Explorer navigator, click <http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl>.
  - ii. In the WSDL Details window, under **Other Actions**, click **Launch Web Service Wizard** > leave **Web Service Client** selected > **Go**.
  - iii. On the first page of the wizard, click the **Client project** link in the configuration > Select the **StockQuoteAxisClient** client project > **OK**.
 

**Tip:** The wizard window might lose focus. You must bring it back into focus manually.
  - iv. The Web service run time must be Apache Axis to generate a JAX-RPC client.
  - v. Click **Finish**.
  - vi. Change the static URL of the service to point to the WebSphere MQ transport for SOAP address for the StockQuoteAxis service. You might choose to skip this step, until you have tested the client with an HTTP server.
    1. Open `StockQuoteAxisServiceLocator.java` and find the declaration for `StockQuoteAxis_address`.
    2. Change the URL to
 

```
"jms:/queue?destination=REQUESTAXIS
&amp;initialContextFactory=com.ibm.mq.jms.NoJndi
&amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
```

**Tip:** Eclipse automatically transforms `&` to `&amp;`, and the reverse, when you copy and paste strings into `.java` code.
- d. Create three Java client classes, each with a main method:
  - i. Create package. Right click **StockQuoteAxisClient/src** > **New Package**. Name it `soap.client` > **Finish**.
  - ii. Select `soap.client` > **New** > **Class**. Name the class `SQASStaticClient` > Select **public static void main(string [] args)** > **Finish**
  - iii. Repeat the procedure to create `SQADynamicClient.java` and `SQADIIClient.java`
- e. Write the client code.
 

[Figure 4](#) through [Figure 8](#) provide examples of the three styles of client code. The examples use an HTTP URL to test the client using the StockQuoteAxis service deployed to an HTTP server. To run the clients against the StockQuoteAxis service deployed using WebSphere MQ transport for SOAP, change the URL to:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.NoJndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

  - [Figure 4](#) and [Figure 6](#) use the proxy generated by Eclipse, which has the extra `StockQuoteAxisproxy` helper class that makes coding a little easier.
  - [Figure 5](#) and [Figure 7](#) use the proxy generated by **amqwdeployWMQService**.
  - [Figure 8](#) uses no proxy classes.

Each of the clients call `com.ibm.mq.soap.Register.extension()` to link to the WebSphere MQ transport for SOAP. The extension is registered in the client deployment descriptor. Client deployment to Axis 1.4 is described in [Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#).
- f. Run the clients by sending the SOAP request to StockQuoteAxis hosted by the WebSphere Application Server Community Edition server configured in the workspace.
  - i. Check that the server is running, StockQuoteAxis is deployed and synchronized.
  - ii. Select or open the client you want to test > Click **Run** in the action bar. Alternatively, click the green Run icon or right click the client in the navigator > **Run As** > **Run Configurations ....** Configure the parameters you require to run the client.
- g. Run the client using the WebSphere MQ transport for SOAP.
 

The procedure uses **amqwdeployWMQService** to deploy the service, and only works with the client that uses the WSDL or proxies built by **amqwdeployWMQService**. To run the client using the original WSDL, or proxies built by eclipse, deploy the service with its deployment descriptor built by Eclipse. Manually start **SimpleJavaListener** using the service port binding name as the `targetServiceName`.

  - i. Follow the instructions in [Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using amqwdeployWMQService](#) to deploy the service to the WebSphere MQ Simple Java SOAP listener. The service deployment works only for the client using the WSDL or client proxies built by **amqwdeployWMQService**.
  - ii. In a command window, run **amqwclientconfig** to create the client deployment descriptor file, `client-deploy.wsdd`.
  - iii. Import `client-deploy.wsdd` into the root of the Java project you want to test using WebSphere MQ transport for SOAP.
    1. Right click the Java project **StockQuoteAxisEclipseClient** > **Import** > **File system** > **Next** > **Browse...**
    2. Browse to the directory containing `client-deploy.wsdd` > **Open** > Select the directory in the left pane of the Import wizard page > Select `client-deploy.wsdd` in the right pane.
    3. Verify **Into folder:** has `StockQuoteAxisEclipseClient` entered > **Finish**.
  - iv. Confirm that the working directory for running a Java application in this project is the `StockQuoteAxisEclipseClient` directory:
 

Right click the Java project **StockQuoteAxisEclipseClient** > **Run as....** > **Run Configurations...** > Select the **(x)= Arguments** tab > Verify that in Working Directory the **Default** radio button is checked, and the path is `StockQuoteAxisEclipseClient`. Alternatively make one of the following choices to select a different location or file containing the client configuration:

    - Select **Other:** > type a directory path of your choice.
    - In the VM arguments window, type `-Daxis.ClientConfigFile=full path to client deployment descriptor file`
  - v. Make sure the URL is configured to point to the service deployed using WebSphere MQ transport for SOAP. Run the client as described in step [ii](#).

**Tip:** Typically, you might encounter one of these errors:

  - i. Exception: No client transport named 'jms' found!.

- ii. A JMS connection error.
- iii. Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java
- iv. Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis

**Explanations:**

- i. client-config.wsdd is not found, or does include the line <transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/> in client-config.wsdd.
- ii. Possibly a build path problem - not including the .jar files in *WebSphere MQ install directory/java/lib*.
- iii. Service deployment problem, either with server-config.wsdd, or with parameters passed to **SimpleSoapListener**.
- iv. Mismatch between deployment descriptor and the implementation of the service.

If you are having difficulty running the client in Eclipse, try using a command window:

- i. Switch to the StockQuoteAxisEclipseClient\bin directory in the workspace directory tree.
- ii. Run **amqwsetcp** and **amqwclientconfig**
- iii. Run `java soap/client/SQASstaticClient`.

### Sample JAX-RPC Web service clients

The sample Java Web service clients shipped with WebSphere MQ are installed in *WMQ install directory\tools\soap\samples\java\clients*.

#### **SQAxis2Axis.java**

SQAxis2Axis.java, [Figure 1](#), is a dynamic proxy client to invoke the StockQuoteAxis service. You can override the URL of the service, which is compiled into the dynamic proxy, by providing a URL on the command line.

#### **SQAxis2DotNet.java**

SQAxis2DotNet.java, [Figure 2](#), is a dynamic proxy client to invoke the StockQuoteDotNet service. You can override the URL of the service, which is compiled into the dynamic proxy, by providing a URL on the command line.

#### **WsdClient.java**

WsdClient.java, [Figure 3](#), is a dynamic invocation client to invoke either the StockQuoteDotNet or StockQuoteAxis service. The client invokes the StockQuoteAxis service by default. Add the command-line option -D to invoke the StockQuoteDotNet service and -w to provide a different port to the port in .\generated\StockQuoteDotNet\_Wmq.wsdl

*Figure 1. SQAxis2Axis.java*

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
    public static void main(String[] args) {
        com.ibm.mq.soap.Register.extension();
        try {
            StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
            StockQuoteAxis service = null;
            if (args.length == 0)
                service = locator.getSoapServerStockQuoteAxis_Wmq();
            else
                service = locator.getSoapServerStockQuoteAxis_Wmq(
                    new java.net.URL(args[0]));
            System.out.println("Response: " + service.getQuote("XXX"));
        } catch (Exception e) {
            System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
            e.printStackTrace();
        }
        System.exit(2);
    }
}
```

*Figure 2. SQAxis2DotNet.java*

```
public class SQAxis2DotNet {
    public static void main(String[] args) {
        com.ibm.mq.soap.Register.extension();
        try {
            StockQuoteDotNet locator = new StockQuoteDotNetLocator();
            StockQuoteDotNetSoap_PortType service = null;
            if (args.length == 0)
                service = locator.getStockQuoteDotNetSoap();
            else
                service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                    args[0]));
            System.out.println("Response: " + service.getQuoteDOC("XXX"));
        } catch (Exception e) {
            System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
            e.printStackTrace();
        }
        System.exit(2);
    }
}
```

*Figure 3. WsdClient.java*

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdClient {
```

```

public static void main(String[] args) {
    String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
    try {
        Register.extension();
        Options opts = new Options(args);
        if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
        } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
        }
        if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
        System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
            + " resolving uri to ...");
        QName servQN = new QName(namespace, wsdlService);
        QName portQN = new QName(namespace, wsdlPort);
        Service service = ServiceFactory.newInstance().createService(
            new URL(wsdlSource), servQN);
        Call call = (Call) service.createCall(portQN, "getQuote");
        wsdlTargetURI = call.getTargetEndpointAddress().toString();
        System.out.println(" " + wsdlTargetURI + " !");
        Object ret = call.invoke(new Object[] { "XXX" });
        System.out.println("Response: " + ret);
    } catch (Exception e) {
        System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
        e.printStackTrace();
        System.exit(2);
    }
}
}
}

```

The example clients used in this task:

*Figure 4. Static client using Eclipse generated proxy*

```

package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
    public static void main(String[] args) {
        try {
            com.ibm.mq.soap.Register.extension();
            StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
            System.out.println("Static client synchronous result is:"
                + sqa.getQuote("ibm"));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

```

*Figure 5. Static client using amqwdeployWMQService generated proxy*

```

package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
    public static void main(String[] args) {
        try {
            com.ibm.mq.soap.Register.extension();
            StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
            StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
            System.out.println("Static client synchronous result is: "
                + sqa.getQuote("ibm"));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

```

*Figure 6. Dynamic client using Eclipse generated proxy*

```

package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
    public static void main(String[] args) {
        try {
            com.ibm.mq.soap.Register.extension();
            StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
                "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
            System.out.println("Dynamic client synchronous result is: "
                + sqa.getQuote("ibm"));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

```

*Figure 7. Dynamic client using amqwdeployWMQService generated proxy*

```

package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;

```

```

public class SQAStaticClient {
    public static void main(String[] args) {
        try {
            com.ibm.mq.soap.Register.extension();
            URL sqURL = new URL(
                "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
            StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
            StockQuoteAxis sq = locator.getSoapServerStockQuoteAxis_Wmq(sqURL);
            System.out.println("Dynamic client synchronous result is: "
                + sq.getQuote("ibm"));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

```

Figure 8. DII client (No proxy)

```

package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQAStaticClient {
    public static void main(String[] args) {
        try {
            com.ibm.mq.soap.Register.extension();
            URL wsdl = new URL(
                "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
            Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
                new QName("http://server.soap", "StockQuoteAxisService"));
            Call SQAcall = SQAService.createCall(new QName("http://server.soap",
                "StockQuoteAxis"), "getQuote");
            System.out.println("DII client synchronous result is "
                + SQAcall.invoke(new Object[] { "ibm" }));
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }
}

```


Parent topic: [Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP](#)

#### Related tasks

[Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)  
[Developing a .NET 1 or 2 client for WebSphere transport for SOAP using Microsoft Visual Studio 2008](#)  
[Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)

#### Related information

[Eclipse Galileo Packages](#)

 This build: January 26, 2011 11:09:46

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20534\_

## 1.5.2. Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse

Develop an Axis2 Web service client to run using the WebSphere® MQ transport for SOAP. The sample Axis2 clients provided with WebSphere MQ transport for SOAP are listed, and the **wsimport** command used to generate proxies.

### Before you begin

Obtain the Axis2 libraries, and configure a development and test environment to run the client.

**Note:** The naming of versions and releases used by Axis causes confusion. Typically, Axis 1.4 refers to the JAX-RPC implementation, and Axis2 to the JAX-WS implementation.

Axis 1.4 is a version level. If you search for Axis 1.4 on the internet, you are taken to <http://ws.apache.org/axis/>. The page contains a list of preceding versions of Axis (1.2, 1.3) and the April 22, 2006, final release of Axis 1.4. There are later releases of Axis 1.4, that fix bugs, but they are all known as Axis 1.4. It is one of these bug fix releases that is shipped with WebSphere MQ. For Axis 1.4, use the version of `axis.jar` that is shipped with WebSphere MQ rather than the one obtainable from <http://ws.apache.org/axis/>.

The Axis Web site also refers to Axis 1.1 to refer to all the versions of what is more typically called Axis 1.4. Axis 1.2 is used to refer to what is typically called Axis2.

Axis 1.5 is not a later release of Axis 1.4, it is an Axis2 release. If you search for Axis 1.5 you are directed to <http://ws.apache.org/axis2/>. <http://ws.apache.org/axis2/download.cgi> contains a list of release versions of Axis2, labeled 0.9 to 1.5.1 (and including, confusingly version 1.4). The release version of Axis2 to use with WebSphere MQ transport for SOAP is 1.4.1. Download Axis2 1.4.1 from [http://ws.apache.org/axis2/download/1\\_4\\_1/download.cgi](http://ws.apache.org/axis2/download/1_4_1/download.cgi).

You can choose to generate proxies for the Web service clients for WebSphere MQ transport for SOAP using either **wsimport** or the tooling provided with an IDE. Eclipse IDE for Java EE Developer 3.5 SR1 uses **wsdl2java.wsimport** is supplied with Java 6. You can use Java 5 to run client proxies generated either with **wsimport** or **wsdl2java**.

The sample Web service Axis2 clients provided with WebSphere MQ transport for SOAP were developed using **wsimport**; see [Sample Axis2 clients](#).

The task that follows demonstrates how to generate and use the proxies produced by the Web services wizard that is packaged with Eclipse IDE for Java EE Developers. The sample clients show how to use the proxies produced by **wsimport**.

To use the Web services wizard, you must add an application server that supports Axis2 to the workbench. The steps show how to configure WASCE to support Axis2 using the workbench.

1. Configure the application server used in Eclipse IDE for Java EE Developers to support Axis2. In this example, configure the WASCE 2.1.4, application server, which is part of the workspace created in [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using](#)



Eclipse.

- a. Open the workspace preferences to configure the server: Open **Window > Preferences**.
  - b. Check the installed JRE is Java50: Click **Installed JREs**.
  - c. Add WASCE as the server: Click **Server > Runtime environments > Add... > IBM > WASCE v2.1** > Next. The JRE must be Java50 > Browse to the WASCE installation directory > **OK > Finish**. You must have installed the WASCE plug-in for Eclipse Java EE IDE for Web Developers.
  - d. Add Axis2: Click **Web Services > Axis2 Preferences**. On the Axis2 Runtime tab > **Browse...** Open the directory containing the many *Axis2* jar files > **Apply**.
  - e. Associate WASCE with Axis2: Click **Web Services > Server and Runtime**. Under **Server** select **IBM WASCE v2.1 Server**, and under **Web service runtime**, select **Apache Axis2 > Apply > OK**
  - f. Start the server: Open the Web perspective and open the Servers view. Right click in the Servers view > **New > Server. IBM WASCE v2.1 Server** is selected and configured > **Finish**. Start the server.
2. Check that you have deployed the StockQuoteAxis service to WASCE to run the Web service wizard.
  3. To test the service with the WebSphere MQ transport for SOAP service, deploy the service to a WebSphere MQ transport for SOAP listener for Axis 1.4; see [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#).

**About this task**

The Eclipse IDE for Java EE Developers uses Java50 and the Web services wizard to generate the proxy classes for the service. The proxy classes are different to the classes created by the **wsimport** tool provided with Java 6. An alternative approach is to generate the proxy classes using **wsimport** and import the packages it creates into your Eclipse Java EE IDE for Web Developers.

The Web services wizard in the Eclipse IDE for Java EE Developers builds a Web service client in a Web project. You can run the client as a simple Java application; it does not require an application server. You can also transfer the code to a Java project, and configure the build path to include the Axis2 JAR files.

**Procedure**

1. Create a Web project in a new Enterprise project:
  - a. With nothing selected in the Project Explorer > Right-click the white space > **New > Enterprise Application Project** > Name it *StockQuoteAxis2EAR* > **Finish**. Reply **No** to the window giving you the option of opening the Java EE perspective. The defaults are set to use WASCE.
  - b. Right-click *StockQuoteAxis2EAR* > **New > Dynamic Web Project**. Name the project *StockQuoteAxis2WebClient* > Check the EAR membership box to add the project to **StockQuoteAxis2EAR**. WASCE 2.1 is selected as the Target runtime.
  - c. In the Configuration section of the New Dynamic Web Project page > **Modify...** > Check the Axis2 Web services project facet. **Dynamic Web Module 2.5, Java 5.0**, and **WASCE deployment 1.2** are already checked. > **OK > Finish**. Reply **No** to the window giving you the option of opening the Java EE perspective.
2. Add the Axis2 classes to the build path.
  - a. Right-click the project *StockQuoteAxis2WebClient* > **Build Path > Configure Build Path...**
  - b. Click **Add External JARs** and browse to the folder where you have unpacked Axis2. Select all the JAR files in *Axis2 folder\lib* > **OK**.
3. Import WSDL for the service into the workspace and generate the client proxy:
 

In this example, the WSDL document contains the HTTP service binding and becomes the target for the static Web client proxy. You can modify the URL in the Web service binding to point to the WebSphere MQ transport for SOAP URL before generating the client proxy. The static Web client proxy is then the service that is deployed to WebSphere MQ transport for SOAP.

  - a. Launch the Web Services Explorer: either use the icon in the action bar, or **Run > Launch the Web Services Explorer**.
  - b. Select the WSDL explorer by clicking the WSDL icon in the top right of the Web Services Explorer window > Click **WSDL Main** in the Navigator window > Type the URL of the StockQuoteAxis WSDL file > **Go**. In this example, obtain the WSDL directly from the HTTP service: `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`
  - c. In the Navigator, click the line with the URL of the Web service. In the WSDL Details window, under **Other Actions**, click **Import WSDL to Workbench** > Select a **StockQuoteAxis2WebClient** as the **Workbench project** > Type the **WSDL file name**, *StockQuoteAxisHTTP.wsdl* > **Go**.
  - d. Right-click **StockQuoteAxisHTTP.wsdl** > **Web Services > Generate Client**. Check the configuration information about the Web services page of the wizard is as follows: Server: IBM@ WASCE v2.1 Server, Web service runtime: Apache Axis2, Client project: *StockQuoteAxis2WebClient*, Client EAR project: *StockQuoteAxisEAR*. To correct the configuration, click the lines that are wrong.
  - e. Click **Next** > verify the code generation settings > **Finish**. Notice that a new package, *soap.server*, is created and it contains the proxies you require.
4. Configure the project to run WebSphere MQ transport for SOAP as the JMS transport. WebSphere MQ transport for SOAP provides a *transportSender*, but no *transportReceiver*. In other words, WebSphere MQ transport for SOAP supports Axis2 clients. Currently it does not support Axis2 services.
  - a. In the **StockQuoteAxis2WebClient** project, right-click `WebContent\WEB-INF\conf\axis2.xml` > **Open with... > XML editor**.
  - b. Search for the last *transportSender* (towards the end of the file) and find the commented out JMS *transportSender* > Right-click the line > **Add before... > transportSender**.
  - c. Right-click **transportSender** > **Add Attribute > Name** > Right-click **transportSender** > **Add Attribute > Class**.
  - d. Right-click **Name** > **Edit Attribute** > Type the **Value**: `jms`
  - e. Right-click **Class** > **Edit Attribute** > Type the **Value**: `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`. > **Save**.
  - f. Add `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` to the build path: Right-click **StockQuoteAxis2WebClient** > **Build Path > Configure Build Path...** > Click the Libraries tab > **Add External JARs...** Select all the JARs in `WMQ Install root\java\lib` > **OK**.
5. Create a synchronous static client, test it using HTTP, then convert the proxy to run the static client using WebSphere MQ transport for SOAP.
  - a. Right-click **Java Resources: src** > **New > Package** > Name the package *soap.client* > **Finish**
  - b. Right-click **soap.client** > **New > Class** > Name the class *SQA2StaticClient* > **Finish**.
  - c. Replace the class with the code in [Figure 1](#) > **Save**.

Figure 1. *SQA2StaticClient.java*

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
```

```

public static void main(String[] args) {
    try {
        StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
        GetQuote request = new GetQuote();
        request.setSymbol("ibm");
        System.out.println("Response is: "
            + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
    }
}
}

```

6. Test the client with the StockQuoteAxis service deployed to WASCE, and with WebSphere MQ transport for SOAP.

- a. In the Project Explorer, right-click **SQA2StaticClient** > **Run as...** > **Java Application**. The result, `Response is 55.25`, appears in the Console view. You can also select the WASCE console window in the Console view, and see the output on the WASCE server, `StockQuoteAxis` called with parameter: `ibm`.
- b. The proxy was built with the service address, `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis`, and so the static client calls the service running on HTTP. You can change the static client to call the service using WebSphere MQ transport for SOAP. The following instructions change the service address in `StockQuoteAxisServiceStub.java` without rebuilding the proxy, and configure the `SQA2StaticClient` runtime parameters to load `axis2.xml`. You configure `axis2.xml` to use WebSphere MQ transport for SOAP.
- c. Open `StockQuoteAxisServiceStub.java` > Replace the two occurrences of `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` with,

```

jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE

```

If you run `SQA2StaticClient` now, it throws an exception because it has not found a `transportSender` configured for JMS or because the transport sender does not support addressing. You need to make two more changes to the configuration:

- d. In `axis2.xml` find the following setting, and comment it out as it says:

```

<!-- Comment this to disable Addressing -->
<module ref="addressing"/>

```

- e. In the Project Explorer, right-click **SQA2StaticClient** > **Run as...** > **Run Configurations...** Switch to the **(x)= Arguments** tab, and in the **VM arguments** input area, type the path to the `axis2.conf` file > **Apply** > **Run**. The VM argument is: `-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml`. Or you can provide a standard path to the `Axis2` configuration file.
- f. Run `SQA2StaticClient` again. On this run, you are using the WebSphere MQ transport for SOAP. Confirm it by checking there is no new output in the WASCE console. Open the console or command window that is associated with `SimpleJavaListener`, and look for the output from `StockQuoteAxis` called with parameter: `ibm`.

7. Create a dynamic client for HTTP and WebSphere MQ transport for SOAP, and test it.

- a. Right-click **soap.client** > **New** > **Class** > Name the class `SQA2DynamicClient` > **Finish**.
- b. Replace the class with the code in [Figure 2](#) > **Save**.

*Figure 2. SQA2DynamicClient.java*

```

package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
    public static void main(String[] args) {
        try {
            StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
                "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
            GetQuote request = new GetQuote();
            request.setSymbol("ibm");
            System.out.println("HTTP Sync: "
                + (stub.getQuote(request)).getGetQuoteReturn());
            stub = new StockQuoteAxisServiceStub(
                "jms:/queue?destination=REQUESTAXIS@QM1"
                + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
                + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
            System.out.println("JMS sync: "
                + (stub.getQuote(request)).getGetQuoteReturn());
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

- c. Create a Run configuration for `SQA2DynamicClient.java`, and add the path to `axis2.xml`: `-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml`
- d. Run `SQA2DynamicClient`. Check the console output for the `SQA2DynamicClient`, WASCE and **SimpleJavaListener**.

8. Create an asynchronous client, and access the result in a callback handler, and in the main program thread.

The asynchronous client proxies created by the Web service wizard for Eclipse Java EE IDE for Web Developers differ from the proxies created by **wsimport**. The **wsimport** proxies use `Future`, `Response`, and `AsyncHandler` generic types.

The Web service wizard for Eclipse Java EE IDE for Web Developers creates a `StockQuoteAxisServiceCallbackHandler` abstract class. You must extend `StockQuoteAxisServiceCallbackHandler` and create a callback handler.

- a. Right-click **soap.client** > **New** > **Class** > Name the class `SQA2CallbackHandler` > **Finish**.
- b. Replace the class with the code in [Figure 3](#).

*Figure 3. SQA2CallbackHandler.java*

```

package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
    private boolean complete = false;
}

```

```

SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
}
public void receiveResultGetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
}
public boolean isComplete() {
    return complete;
}
}
}

```

- c. Right-click **soap.client** > **New** > **Class** > Name the class `SQA2AsyncClient` > **Finish**.
- d. Replace the class with the code in [Figure 4](#).

Figure 4. `SQA2AsyncClient.java`

```

package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
    public static void main(String[] args) {
        try {
            StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
                "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
            GetQuote request = new GetQuote();
            request.setSymbol("ibm");
            System.out.println("HTTP Sync: "
                + (stub.getQuote(request)).getGetQuoteReturn());
            SQA2CallbackHandler callback = new SQA2CallbackHandler();
            stub.startGetQuote(request, callback);
            do {
                System.out.println("Waiting for HTTP callback");
                Thread.sleep(2000);
            } while (!callback.isComplete());
            System.out.println("HTTP poll: "
                + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
            stub = new StockQuoteAxisServiceStub(
                "jms:/queue?destination=REQUESTAXIS@QM1"
                + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.NoJndi"
                + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
            System.out.println("JMS Sync: "
                + (stub.getQuote(request)).getGetQuoteReturn());
            callback = new SQA2CallbackHandler();
            stub.startGetQuote(request, callback);
            while (!callback.isComplete()) {
                System.out.println("Waiting for JMS callback");
                Thread.sleep(2000);
            }
            System.out.println("JMS poll: "
                + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}

```

The console output is listed in [Figure 5](#).

Figure 5. Console output from `SQA2AsyncClient.java`

```

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25

```

### Sample Axis2 clients

The sample proxies are generated using the **wsimport** tool that is packaged with Java 6. Six samples are provided:

1. [DynamicProxyClientSync.java](#)
2. [DynamicProxyClientAsyncPolling.java](#)
3. [DynamicProxyClientAsyncCallback.java](#)
4. [DispatchClientSync.java](#)
5. [DispatchClientAsyncPolling.java](#)
6. [DispatchClientAsyncCallback.java](#)

The client samples are generated for the sample `StockQuoteAxis` server. Generate the WSDL with the **amqwdpoyWMQServer** command, specifying the **-w** switch to select `rpcLiteral` style. Use the following command to generate the proxies for the samples:

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

Figure 6. `DynamicProxyClientSync.java`

```

package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

```

```

public class DynamicProxyClientSync {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DynamicProxyClientSync");

            System.out.println("Creating proxy instance for service StockQuoteAxisService");
            StockQuoteAxisService stub = new StockQuoteAxisService();
            StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

            System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
            service.getQuoteOneWay("48");
            System.out.println(" > getQuoteOneWay has returned");

            System.out.println("Invoking getQuote Request Reply operation synchronously...");
            float result = service.getQuote("48");
            System.out.println(" > getQuote has returned result of " + result);

            System.out.println("End of sample");
        }
        catch (Exception fault) {
            // Identify the cause of the Axis Fault
            System.err.println(fault.toString());
            Throwable e = fault.getCause();
            for (int i = 1; e != null; i++) {
                // The toString method on an MQAxisException will cause the message, explanation and user
                // action.
                System.err.println("Exception(" + i + "): " + e.toString());

                if (e.getCause() != null) {
                    e = e.getCause();
                }
                else {
                    break;
                }
            } // end of for loop
        } // end of catch block
    }
}

```

Figure 7. *DynamicProxyClientAsyncPolling.java*

```

package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DynamicProxyClientAsyncPolling");

            System.out.println("Creating proxy instance for service StockQuoteAxisService");
            StockQuoteAxisService stub = new StockQuoteAxisService();
            StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

            System.out
                .println("Invoking getQuoteAsync Request Reply operation asynchronously by polling...");
            Response<Float> response = service.getQuoteAsync("49");

            /** Sleep main thread until response arrives */
            System.out.println("Waiting for response to arrive...");
            while (!response.isDone()) {
                Thread.sleep(100);
            }
            System.out.println(" > Response received");

            /** Retrieve the result */
            try {
                Float result = response.get();
                System.out.println(" > getQuoteAsync call has returned result of " + result);
            }
            catch (CancellationException ce) {
                // processing was cancelled via response.cancel()
            }

            System.out.println("End of sample");
        }
        catch (Exception fault) {
            // Identify the cause of the Axis Fault
            System.err.println(fault.toString());
            Throwable e = fault.getCause();
            for (int i = 1; e != null; i++) {
                // The toString method on an MQAxisException will cause the message, explanation and user
                // action.
                System.err.println("Exception(" + i + "): " + e.toString());

                if (e.getCause() != null) {
                    e = e.getCause();
                }
                else {
                    break;
                }
            } // end of for loop
        } // end of catch block
    }
}

```

```

    }
}

```

**Figure 8.** *DynamicProxyClientAsyncCallback.java*

```

package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DynamicProxyClientAsyncCallback");

            System.out.println("Creating proxy instance for service StockQuoteAxisService");
            StockQuoteAxisService stub = new StockQuoteAxisService();
            StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

            DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

            System.out
                .println("Invoking getQuoteAsync Request Reply operation asynchronously using a callback...");
            Future<?> monitor = service.getQuoteAsync("50", handler);
            System.out.println(" > Invoke call has returned");

            /** Sleep main thread until handler has been notified **/
            System.out.println("Waiting for handler to be called...");
            while (!monitor.isDone()) {
                Thread.sleep(100);
            }

            System.out.println("End of sample");
        }
        catch (Exception fault) {
            // Identify the cause of the Axis Fault
            System.err.println(fault.toString());
            Throwable e = fault.getCause();
            for (int i = 1; e != null; i++) {
                // The toString method on an MQAxisException will cause the message, explanation and user
                // action.
                System.err.println("Exception(" + i + "): " + e.toString());

                if (e.getCause() != null) {
                    e = e.getCause();
                }
                else {
                    break;
                }
            } // end of for loop
        } // end of catch block
    }

    public void handleResponse(Response<Float> response) {
        try {
            Float result = response.get();
            System.out.println(" > Async Handler has received a result of " + result);
        }
        catch (Exception fault) {
            // Identify the cause of the Axis Fault
            System.err.println("Exception in handleResponse");
            System.err.println(fault.toString());
            Throwable e = fault.getCause();
            for (int i = 1; e != null; i++) {
                // The toString method on an MQAxisException will cause the message, explanation and user
                // action.
                System.err.println("Exception(" + i + "): " + e.toString());

                if (e.getCause() != null) {
                    e = e.getCause();
                }
                else {
                    break;
                }
            } // end of for loop
        } // end of catch block
    }
}

```

**Figure 9.** *DispatchClientSync.java*

```

package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

```

```

public class DispatchClientSync {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DispatchClientSync");

            String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
                + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
                + "&initialContextFactory=com.ibm.mq.jms.NoJndi&targetService=soap.server.StockQuoteAxis.java";

            QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
            QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteAxis_Wmq");

            Service service = Service.create(serviceName);
            service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

            /** Create a Dispatch instance from a service */
            System.out.println("Creating dispatch instance for service StockQuoteAxisService");
            Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
                Service.Mode.MESSAGE);
            System.out.println(" > Dispatch instance created.");

            /*****
             * Create OneWay SOAPMessage request.
             *****/
            MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

            System.out.println("\nCreating a OneWay SOAP Message");
            SOAPMessage request = mf.createMessage();

            /** Obtain the SOAPEnvelope and header and body elements */
            SOAPPart part = request.getSOAPPart();
            SOAPEnvelope env = part.getEnvelope();
            SOAPHeader header = env.getHeader();
            SOAPBody body = env.getBody();

            /** Construct the message payload */
            SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
                "soap.server.StockQuoteAxis_Wmq");
            SOAPElement value = operation.addChildElement("in0");
            value.addAttribute(new QName("http://www.w3.org/2001/XMLSchema-instance", "type"), "string");
            value.addTextNode("XXX");
            request.saveChanges();
            System.out.println(" > SOAP Message created.");

            /** Invoke the service endpoint */
            System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
            dispatch.invokeOneWay(request);
            System.out.println(" > getQuoteOneWay call has returned");

            /*****
             * Create Request Reply SOAPMessage request.
             *****/
            mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

            System.out.println("\nCreating a Request Reply SOAP Message");
            request = mf.createMessage();

            /** Obtain the SOAPEnvelope and header and body elements */
            part = request.getSOAPPart();
            env = part.getEnvelope();
            header = env.getHeader();
            body = env.getBody();

            /** Construct the message payload */
            operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
            value = operation.addChildElement("in0");
            value.addAttribute(new QName("http://www.w3.org/2001/XMLSchema-instance", "type"), "string");
            value.addTextNode("XXX");
            request.saveChanges();
            System.out.println(" > SOAP Message created.");

            /** Invoke the service endpoint */
            System.out.println("Invoking getQuote Request Reply operation synchronously...");
            SOAPMessage ans = dispatch.invoke(request);
            System.out.println(" > getQuote call has returned");

            /** Retrieve the result */
            part = ans.getSOAPPart();
            env = part.getEnvelope();
            body = env.getBody();

            /** Define name of the SOAP folders we are interested in */
            QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
            QName resultName = new QName("getQuoteReturn");

            /** Retrieve result from SOAP envelope */
            System.out.println("Parsing SOAP response...");
            SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
            SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultName).next();
            String message = responseElement.getValue();
            System.out.println(" > Response contains result of " + message);

            System.out.println("End of sample");
        }
        catch (Exception fault) {
            // Identify the cause of the Axis Fault
            System.err.println(fault.toString());
            Throwable e = fault.getCause();
        }
    }
}

```

```

for (int i = 1; e != null; i++) {
    // The toString method on an MQAxisException will cause the message, explanation and user
    // action.
    System.err.println("Exception(" + i + "): " + e.toString());

    if (e.getCause() != null) {
        e = e.getCause();
    }
    else {
        break;
    }
} // end of for loop
} // end of catch block
}
}

```

Figure 10. DispatchClientAsyncPolling.java

```

package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DispatchClientAsyncPolling");

            String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
                + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
                + "&initialContextFactory=com.ibm.mq.jms.NoJndi&targetService=soap.server.StockQuoteAxis.java";

            QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
            QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteAxis_Wmq");

            Service service = Service.create(serviceName);
            service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

            /** Create a Dispatch instance from a service. */
            System.out.println("Creating dispatch instance for service StockQuoteAxisService");
            Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
                Service.Mode.MESSAGE);
            System.out.println(" > Dispatch instance created.");

            /** Create SOAPMessage request. */
            MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

            System.out.println("Creating a Request Reply SOAP Message");
            SOAPMessage request = mf.createMessage();

            /** Obtain the SOAPEnvelope and header and body elements */
            SOAPPart part = request.getSOAPPart();
            SOAPEnvelope env = part.getEnvelope();
            SOAPHeader header = env.getHeader();
            SOAPBody body = env.getBody();

            /** Construct the message payload */
            SOAPElement operation = body.addChildElement("getQuote", "ns1",
                "soap.server.StockQuoteAxis_Wmq");
            SOAPElement value = operation.addChildElement("in0");
            value.addAttribute(new QName("http://www.w3.org/2001/XMLSchema-instance", "type"), "string");
            value.addTextNode("XXX");
            request.saveChanges();
            System.out.println(" > SOAP Message created.");

            /** Invoke the service endpoint */
            System.out.println("Invoking getQuote Request Reply operation asynchronously by polling...");
            Response<SOAPMessage> response = dispatch.invokeAsync(request);
            System.out.println(" > getQuote call has returned");

            /** Sleep main thread until response arrives */
            System.out.println("Waiting for response to arrive...");
            while (!response.isDone()) {
                Thread.sleep(100);
            }
            System.out.println(" > Response received");

            /** retrieve the result */
            SOAPMessage ans = response.get();
            part = ans.getSOAPPart();
            env = part.getEnvelope();
            body = env.getBody();

            /** Define name of the SOAP folders we are interested in */
            QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
            QName resultName = new QName("getQuoteReturn");

            /** Retrieve result from SOAP envelope */
            SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();

```

```

SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultName).next();
String message = responseElement.getValue();
System.out.println("> Response contains result of " + message);

System.out.println("End of sample");

}
catch (Exception fault) {
    // Identify the cause of the Axis Fault
    System.err.println(fault.toString());
    Throwable e = fault.getCause();
    for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
            e = e.getCause();
        }
        else {
            break;
        }
    } // end of for loop
} // end of catch block
}
}

```

Figure 11. DispatchClientAsyncCallback.java

```

package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

    public static void main(String[] args) {
        try {
            System.out.println("Starting sample DispatchClientAsyncCallback");

            String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
                + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
                + "&initialContextFactory=com.ibm.mq.jms.NoJndi&targetService=soap.server.StockQuoteAxis.java";

            QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
            QName portName = new QName("soap.server.StockQuoteAxis_Wmq", "soap.server.StockQuoteAxis_Wmq");

            Service service = Service.create(serviceName);
            service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

            /** Create a Dispatch instance from a service. */
            System.out.println("Creating dispatch instance for service StockQuoteAxisService");
            Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
                Service.Mode.MESSAGE);
            System.out.println("> Dispatch instance created.");

            /** Create SOAPMessage request. */
            MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

            System.out.println("Creating a Request Reply SOAP Message");
            SOAPMessage request = mf.createMessage();

            /** Obtain the SOAPEnvelope and header and body elements */
            SOAPPart part = request.getSOAPPart();
            SOAPEnvelope env = part.getEnvelope();
            SOAPHeader header = env.getHeader();
            SOAPBody body = env.getBody();

            /** Construct the message payload. */
            SOAPElement operation = body.addChildElement("getQuote", "ns1",
                "soap.server.StockQuoteAxis_Wmq");
            SOAPElement value = operation.addChildElement("in0");
            value.addAttribute(new QName("http://www.w3.org/2001/XMLSchema-instance", "type"), "string");
            value.addTextNode("XXX");
            request.saveChanges();
            System.out.println("> SOAP Message created.");

            /** Invoke the service endpoint. */
            DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

            System.out
                .println("Invoking getQuote Request Reply operation asynchronously using a callback...");
            Future<?> monitor = dispatch.invokeAsync(request, handler);
            System.out.println("> getQuote call has returned");

            /** Sleep main thread until handler has been notified */
            System.out.println("Waiting for handler to be called...");

```



```

while (!monitor.isDone()) {
    Thread.sleep(100);
}

System.out.println("End of sample");
}
catch (Exception fault) {
    // Identify the cause of the Axis Fault
    System.err.println(fault.toString());
    Throwable e = fault.getCause();
    for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
            e = e.getCause();
        }
        else {
            break;
        }
    } // end of for loop
} // end of catch block
}

public void handleResponse(Response<SOAPMessage> response) {
    try {
        // retrieve the result
        SOAPMessage ans = response.get();
        SOAPPart part = ans.getSOAPPart();
        SOAPEnvelope env = part.getEnvelope();
        SOAPBody body = env.getBody();

        /** Define name of the SOAP folders we are interested in */
        QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
        QName resultName = new QName("getQuoteReturn");

        /** Retrieve result from SOAP envelope */
        SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
        SOAPElement responseElement = (SOAPElement) bodyElement.getChildElements(resultName).next();
        String result = responseElement.getValue();

        System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
        // Identify the cause of the Axis Fault
        System.err.println("Exception in handleResponse");
        System.err.println(fault.toString());
        Throwable e = fault.getCause();
        for (int i = 1; e != null; i++) {
            // The toString method on an MQAxisException will cause the message, explanation and user
            // action.
            System.err.println("Exception(" + i + "): " + e.toString());

            if (e.getCause() != null) {
                e = e.getCause();
            }
            else {
                break;
            }
        } // end of for loop
    } // end of catch block
}
}
}

```

**Parent topic:** [Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP](#)


#### Related tasks

[Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse](#)  
[Developing a .NET 1 or 2 client for WebSphere transport for SOAP using Microsoft Visual Studio 2008](#)  
[Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)

#### Related information

[Eclipse Galileo Packages](#)

<sup>1</sup> Add the line: `System.out.println("StockQuoteAxis called with parameter: " + symbol);` to `getQuote` in `StockQuoteAxis.java` if you have not modified the sample already. Restart `SimpleJavaListener`.

 This build: January 26, 2011 11:09:48

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20535\_

### 1.5.3. Developing a .NET 1 or 2 client for WebSphere transport for SOAP using Microsoft Visual Studio 2008

Develop an .NET 1 or 2 Web service client to run using the WebSphere® MQ transport for SOAP.

#### Before you begin

You can start the development of a .NET 1 or 2 client in a number of different ways:

1. Use **amqwdployWMQService** to generate client stubs from a Web service and import them into Visual Studio.

2. Use **java2wsdl** to generate WSDL from a Java implementation of a Web service, and then use `wsdl.exe`, which is shipped with .NET, to generate client stubs.
3. Generate WSDL from a .NET `.asmx` implementation of the service using **amqswsdl**, and then use `wsdl.exe`.
4. If you have developed and deployed the service for HTTP, use the **Add Web reference...** wizard in Visual studio to configure the client to access the HTTP service. Alter the URL to refer to the service deployed to WebSphere MQ transport for SOAP.

The task uses the service developed in [Developing a .NET 1 or 2 service for WebSphere MQ transport for SOAP using Microsoft Visual Studio 2008](#).

### About this task

Follow these steps to create a .NET 1 or 2 Client for HTTP and WebSphere MQ transport for SOAP.

### Procedure

1. Create the client console application and modify it to invoke the StockQuote HTTP Web service.
  - a. Right click **Solution 'StockQuoteDotNet'** in the Solution Explorer > Add... > New Project. Select the **C# Project Type, .NET Framework 2.0, and Console Application**. Name the project `StockQuoteClientDotNet` > **OK**
  - b. Right click **Solution 'StockQuoteDotNet'** in the Solution Explorer > Add... > New Project. Select the **C# Project Type, .NET Framework 2.0, and Console Application**. Name the project `StockQuoteClientDotNet` > **OK**
  - c. Right-click **StockQuoteClientDotNet** > **Set as Startup project**.
  - d. Right-click **StockQuoteClientDotNet** > **Add Web Reference...** > Browse to Web services in this solution > Select **StockQuote** > **Add Reference**. Notice you have added a Web reference to local host and a new configuration file `app.config`.
  - e. In the Solution Explorer, change the name of the console application from `Program.cs` to `StockQuoteClientDotNet.cs` > Click **OK** to changing all the usages of `Program.cs` to `StockQuoteClientDotNet.cs`.
  - f. Replace the contents of `StockQuoteClientDotNet.cs` with the code in [Figure 1](#).

Figure 1. HTTP StockQuoteClientDotNet program

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
    class StockQuoteClientDotNet {
        static void Main(string[] args) {
            try {
                StockQuote stockobj = new StockQuote();
                Console.WriteLine("http reply is: "
                    + stockobj.getNonInlineQuote("http request"));
            }
            catch (System.Exception e) {
                Console.WriteLine("Exception thrown: " + e.ToString());
            }
            Console.ReadLine();
        }
    }
}
```

- g. Launch `StockQuoteClientDotNet` to test against the `StockQuote.asmx` service:
  - i. Press **F5**, click the green arrow in the action bar, or **Debug > Start Debugging (F5)**. If the `StockQuoteDOTNet` project is in the same solution, it starts automatically. Otherwise you need to start the service first. The command window with the results opens behind the workspace. The `Console.ReadLine();` statement prevents it from closing until you press **Enter**.
 

**Tip:** Make sure `StockQuote.asmx` is the Start page in the `StockQuoteDotNet` project.

2. Modify `StockQuoteClientDotNet` to call the `StockQuote.asmx` service using WebSphere MQ transport for SOAP.

- a. Add the lines shown in bold to the client.
 

Figure 2. Modified StockQuoteClientDotNet program

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
    class StockQuoteClientDotNet {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                StockQuote stockobj = new StockQuote();
                Console.WriteLine("http reply is: "
                    + stockobj.getNonInlineQuote("http request"));
                stockobj.Url = "jms:/queue?"
                + "initialContextFactory=com.ibm.mq.jms.NoJndi"
                + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
                + "&targetService=StockQuote.asmx";
                Console.WriteLine("jms reply is: "
                    + stockobj.getNonInlineQuote("jms request"));
            }
            catch (System.Exception e) {
                Console.WriteLine("Exception thrown: " + e.ToString());
            }
            Console.ReadLine();
        }
    }
}
```

Alternatively, modify the default URL. Open **StockQuoteClientDotNet** > **Properties** > **Settings.settings** and change the value of the `StockQuoteClientDotNet_localhost_StockQuote` property to the WebSphere MQ transport for SOAP URL.

- b. Add a reference to `amqsoap.dll`
  - i. In the **StockQuoteClientDotNet** project in the Solution Explorer, right-click **References** > **Add Reference...** > Click the **Browse** tab > browse to `WMQ Install directory\bin` > Select **amqsoap.dll** > **OK**.
3. Test the client with the `StockQuote.asmx` service using WebSphere MQ transport for SOAP.
  - a. Open a command window in the `StockQuoteDotNet` project directory: `.\StockQuoteDotNet\StockQuoteDotNet > Verify` that `.bin\StockQuoteDotNet.dll` exists. If it does not, rebuild the solution.
  - b. Type the command **amqwRegisterdotNet**. You need only run **amqwRegisterdotNet** once per installation.
  - c. If you have run **amqwdeployWMQServer** with the `genAsmxWMQBits`, run the .NET SOAP Listener:

```
generated\server\startWMQListener
```

d. Alternatively run the listener directly:

```
amqwsOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.NoJndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. In Visual Studio 2008, press **F5** to run StockQuoteClientDotNet.

## .NET Framework 1 and .NET Framework 2 Web service clients

The sample .NET clients provided with the WebSphere MQ transport for SOAP use generated stubs to call the sample Axis and .NET services.

For .NET Framework 1 and .NET Framework 2 clients, WebSphere MQ provides access to web services using .NET clients. The **amqwdployWMQService** command has an option, *genProxiestoDotNet*, that generates .NET Framework 1 or .NET Framework 2 client stubs for a Web service. You can also use client stubs generated by the .NET **wsdl** tool, or by Microsoft Visual Studio 2005, or 2008.

The sample .NET Framework 1 and .NET Web service clients are installed in *WMQ install directory\tools\soap\samples\dotnet*.

### SQVB2Axis.vb

SQVB2Axis.vb, [Figure 3](#), is the Visual Basic client to call the **StockQuoteAxisService** service.

### SQVB2DotNet.vb

SQVB2DotNet.vb, [Figure 4](#), Basic client to call the **StockQuoteDotNet** service.

### SQCS2Axis.cs

SQCS2Axis.cs, [Figure 5](#), is the C# client to call the **StockQuoteAxisService** service. You can override the URL of the service by providing a URL on the command line.

### SQCS2DotNet.cs

SQCS2DotNet.cs, [Figure 6](#), is the C# client to call the **StockQuoteDotNet** service. You can override the URL of the service by providing a URL on the command line.

Figure 3. SQVB2Axis

```
Module SQVB2Axis
Function Main(ByVal CmdArgs() As String) As Integer
    IBM.WMQSOAP.Register.Extension()
    Dim obj As New StockQuoteAxisService()
    Dim res As Single = obj.getQuote("fromcs")
    System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
End Function
End Module
```

Figure 4. SQVB2DotNet

```
Module SQVB2DotNet
Function Main(ByVal CmdArgs() As String) As Integer
    IBM.WMQSOAP.Register.Extension()
    Dim obj as new StockQuoteDotNet()
    Dim res as Single = obj.getQuote("fromcs")
    System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
End Function
End Module
```

Figure 5. SQCS2Axis

```
using System;
class SQCS2Axis {
    [STAThread]
    static void Main(string[] args) {
        try {
            IBM.WMQSOAP.Register.Extension();
            StockQuoteAxisService stockobj = new StockQuoteAxisService();
            if (args.GetLength(0) >= 1)
                stockobj.Url = args[0];
            System.Single res = stockobj.getQuote("XXX");
            Console.WriteLine("SQCS2Axis RPC reply is: " + res);
        }
        catch (System.Exception e) {
            Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
                + e.ToString());
        }
    }
}
```

Figure 6. SQCS2DotNet

```
using System;
class SQCS2DotNet {
    [STAThread]
    static void Main(string[] args) {
        try {
            IBM.WMQSOAP.Register.Extension();
            StockQuoteDotNet stockobj = new StockQuoteDotNet();
            if (args.GetLength(0) >= 1)
                stockobj.Url = args[0];
            System.Single res = stockobj.getQuote("XXX");
            Console.WriteLine("RPC reply is: " + res);
            if (args.GetLength(0) == 0) {
                res = stockobj.getQuoteDOC("XXX");
                Console.WriteLine("DOC reply is: " + res);
            }
        }
        catch (System.Exception e) {
            Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
                + e.ToString());
        }
    }
}
```

```


}
}

```

**Parent topic:** [Developing WebSphere MQ Web service clients for WebSphere MQ transport for SOAP](#)

#### Related tasks

[Developing a JAX-RPC client for WebSphere transport for SOAP using Eclipse](#)  
[Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)

 This build: January 26, 2011 11:09:50

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20538\_

## 1.6. Deploying Web services using the WebSphere MQ transport for SOAP

Deploy a Web service to one of a number of different server environments and connect to it using WebSphere® MQ transport for SOAP.

### Before you begin

Develop a Web service and test it using SOAP over HTTP in the target environment.

### About this task

You can deploy a web service to run with WebSphere MQ transport for SOAP in a number of different SOAP run time environments. You can deploy a service to Axis 1.4 using only the software installed with WebSphere MQ. For the other run time environments, you must install additional software.

You are not restricted to running WebSphere MQ transport for SOAP to the servers for which there are deployment instructions. Use the instructions to deploy a service to one of the listed environments.

**Note:** Some integrated environments offer SOAP over JMS using the W3C recommended JMS SOAP binding, as well as the WebSphere MQ transport for SOAP binding. Releases of WebSphere MQ, up to and including 7.0.1.2, support only the WebSphere MQ transport for SOAP binding. From 7.0.1.3 onwards you can deploy Axis2 clients using a URI that conforms to the W3C candidate recommendation for SOAP over JMS. See the tutorial, [Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5](#).

#### [Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using amqwdeployWMQService](#)

Deploy an Axis 1.4 service to WebSphere MQ transport for SOAP by creating a deployment directory, running the **amqwdeployWMQService** command, and starting the Axis 1.4 listener.

#### [Deploying a service to .NET Framework 1 or 2 service to use WebSphere MQ transport for SOAP](#)

Deploy a .NET Framework 1 or 2 service to WebSphere MQ transport for SOAP. Create a deployment directory, run the **amqwdeployWMQService** command, and start the .NET listener.

#### [Deploying a service to CICS Transaction Server to use WebSphere Transport for SOAP](#)

WebSphere MQ transport for SOAP is integrated into CICS Transaction Server 4.1 Web services support.

#### [Deploying a service to WebSphere Application Server to use WebSphere Transport for SOAP](#)

WebSphere MQ transport for SOAP is integrated into the service integration bus on WebSphere Application Server.


#### [Configuring WebSphere Application Server to use W3C SOAP over JMS](#)

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 1 of connecting an Axis2 Web service client and a Web service deployed to WebSphere Application server using the W3C SOAP over JMS protocol. Configure the WebSphere MQ and WebSphere Application Server resources to develop and deploy Web service bound to W3C SOAP over JMS as a transport.

#### [Deploying a service to WebSphere ESB and Process Server service endpoint to use WebSphere Transport for SOAP](#)

WebSphere MQ transport for SOAP is not directly supported by WebSphere ESB and Process Server. You must configure a custom Export.

**Parent topic:** [WebSphere MQ transport for SOAP](#)

 This build: January 26, 2011 11:09:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20400\_

### 1.6.1. Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using amqwdeployWMQService

Deploy an Axis 1.4 service to WebSphere® MQ transport for SOAP by creating a deployment directory, running the **amqwdeployWMQService** command, and starting the Axis 1.4 listener.

#### Before you begin

1. Follow the instructions for installing WebSphere MQ transport for SOAP
2. Verify the installation and your environment using the **runivt** command.
3. To redeploy a service:
  - a. Delete the `./generated` subdirectory, and all its subdirectories.
  - b. Remove requests from the destination queue and delete it.
  - c. Proceed with the instructions from step 2.

## About this task

These instructions are to deploy an Axis 1.4 service for the first time. To restart an Axis 1.4 service, rerun the Axis 1.4 SOAP listener: step [11](#).

Use the following instructions to deploy a new Axis 1.4 service to WebSphere MQ transport for SOAP:

## Procedure

1. Create a directory `deployDir` to hold the deployment files. The deployment utility requires that each service is deployed from a separate directory.
2. Open a command window on Windows, or a command shell using X Window System on UNIX, in `deployDir` to run **amqwdeployWMQService**.
3. Run **amqwsetcp** to set the classpath. The JRE and JDK must be in the classpath, at version 1.4.2 or later, and at the same version level.
4. Copy the class source, `className.java`, into `deployDir`
5. Copy all the Java source files in the same package as `className` into `deployDir/packageName`, where `packageName` is a directory tree corresponding to the package name.
6. Run **javac** `packageName.className`. You might need to add a path to the current directory ".", or to the `packageName` directory for **javac** to find the other classes.
7. Create the Axis WSDL for the service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdll
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. Create the WebSphere MQ resources for the service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

### Tip:

If you want to set up a new queue manager, and the resources it needs, to do development and testing, run **setupWMQSOAP**.

If you want set up the new queue manager as the default, take a copy of **setupWMQSOAP** from the

`WMQ install directory\tools\soap\samples` directory, and add the `-q` parameter to the line

```
call :try -q crtmqm %QMGR%
```

9. Create the Axis listener and deploy the service:

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. If you need to generate the WSDL for the service, generate client stubs, or client proxies, run **amqwdeployWMQService** with one of the following parameters:

- o `genAsmxWsdll`
- o `genAxisWsdll`
- o `genProxiesToDotNet`
- o `genProxiestoAxis`

**Note:** You must generate WSDL before generating the proxies. The `AllAxis` option fails if the `CLASSPATH` is not set up to find all the classes that are imported to compile `className.java`. If there are multiple Java files in the package containing `className.java`, you must compile them first using **javac**. **amqwdeployWMQService** `-f packageName.className.java -c CompileJava` compiles only `className.java`.

11. Start the generated Axis listener.

```
.\generated\server\startWMQJListener.cmd
```


**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

## Related reference

[amqwdeployWMQService: deploy Web service utility](#)

[SimpleJavaListener: WebSphere MQ SOAP Listener for Axis 1.4](#)

[WebSphere MQ SOAP listeners](#)

 This build: January 26, 2011 11:09:41

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20430\_

## 1.6.2. Deploying a service to .NET Framework 1 or 2 service to use WebSphere MQ transport for SOAP

Deploy a .NET Framework 1 or 2 service to WebSphere® MQ transport for SOAP. Create a deployment directory, run the **amqwdeployWMQService** command, and start the .NET listener.

### Before you begin

1. Follow the instructions for installing WebSphere MQ transport for SOAP
2. Verify the installation and your environment using the **runivt** command.
3. The path to the .NET framework files `wsdl.exe` and `csc.exe` must be set. The copies of `wsdl.exe` and `csc.exe` identified by the `PATH` variable must be at the same level of the .NET framework. If you have multiple .NET frameworks installed, or are using Visual Studio, check the `PATH` variable carefully.
4. To redeploy a service:

- a. Delete the `./generated` subdirectory, and all its subdirectories
- b. Remove requests from the destination queue and delete it.
- c. Proceed with the instructions from step 2.

### About this task

These instructions are to deploy a .NET service for the first time. To restart a .NET service, rerun the .NET SOAP listener, step 9.

Use the following instructions to deploy a new .NET Framework 1 or .NET Framework 2 service to WebSphere MQ transport for SOAP:

### Procedure

1. Create a directory `deployDir` to hold the deployment files. The deployment utility requires that each service is deployed from a separate directory.
2. Open a command window in `deployDir` to run **amqwdeployWMQService**.

```
C:\IBM\ID\QuoteClient>
```

3. Run **amqwsetcp** to set the classpath. A classpath is needed only for Axis clients.
4. Copy the .NET service, `className.asmx`, into `deployDir`
5. Build the service implementation into a library (.dll).  
The inline service implementation is in `className.asmx`. The code-behind service implementation might be `className.asmx.cs`.  
[Figure 1](#) shows an example of a command to build a .NET Framework V2 service as a library.  
*Figure 1. Build command for .NET Framework V2 service*

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

6. Copy `className.dll` into `deployDir\bin`.
7. Set up the WebSphere MQ resources, and create the listener required for the service:  

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```
8. If you need to generate the WSDL for the service, generate client stubs, or client proxies, run **amqwdeployWMQService** with one of the following parameters:
  - o `genAsmxWsd1`
  - o `genAxisWsd1`
  - o `genProxiesToDotNet`
  - o `genProxiestoAxis`

**Note:** You must generate WSDL before generating the proxies.


9. Start the generated .NET listener.  

```
.\generated\server\startWMQNListener.cmd
```

**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

### Related reference

[amqwdeployWMQService: deploy Web service utility](#)  
[amqwSOAPNETListener: WebSphere MQ SOAP listener for .NET Framework 1 or 2](#)  
[WebSphere MQ SOAP listeners](#)

 This build: January 26, 2011 11:09:41

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20410\_

## 1.6.3. Deploying a service to CICS Transaction Server to use WebSphere Transport for SOAP

WebSphere® MQ transport for SOAP is integrated into CICS® Transaction Server 4.1 Web services support.

### Before you begin

Use the same tools to develop for a client or service for WebSphere MQ, as you would to develop for HTTP. CICS has tools corresponding to **Java2wsdl** and **wsdl2Java**:

- **DFHWS2LS** takes a Web service description as a starting point. It uses the descriptions of the messages, and the data types used in those messages, to construct high-level language data structures. You can use in the structures in application programs written in different languages.
- **DFHLS2WS** takes a high-level language data structure as a starting point. It uses the structure to construct a Web services description that contains descriptions of messages. It also creates schemas for the messages from the language data structure.

Follow the instructions, [Creating a Web service](#) in the CICS Information Center, to create a Web service.


**About this task**

Follow the instructions, [Configuring CICS to use the WebSphere MQ transport](#) in the CICS Information Center. Using the instructions, you can deploy the Web service to WebSphere MQ transport for SOAP.

**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

**Related information**

[Configuring CICS to use the WebSphere MQ transport](#)

 This build: January 26, 2011 11:09:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20470\_

## 1.6.4. Deploying a service to WebSphere Application Server to use WebSphere Transport for SOAP

WebSphere® MQ transport for SOAP is integrated into the service integration bus on WebSphere Application Server.

**Before you begin**

Use Rational® Application Developer, WebSphere Integration Developer, or a Web services toolkit to develop the Web service.

**About this task**

Use the following instructions to deploy a service using WebSphere MQ transport for SOAP as a SOAP transport on WebSphere Application Server.

**Procedure**

1. Configure WebSphere MQ as the JMS messaging provider for the service integration bus on WebSphere Application Server.
2. Configure the WebSphere MQ resources required by the service.
3. Follow the instructions, [Configuring JMS resources for the synchronous SOAP over JMS endpoint listener](#), in the WebSphere Application Server Network Deployment Information Center. There are corresponding instructions for other WebSphere Application Server platforms.
4. Modify the service URI to conform to the WebSphere MQ transport for SOAP URI.
5. Deploy the service to WebSphere Application Server.


**What to do next**

Deploy the service with HTTP as a transport so that clients can query the service and receive the WSDL in response.

**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

**Related information**

[Configuring JMS resources for the synchronous SOAP over JMS endpoint listener](#)

 This build: January 26, 2011 11:09:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20440\_

## 1.6.5. Configuring WebSphere Application Server to use W3C SOAP over JMS

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 1 of connecting an Axis2 Web service client and a Web service deployed to WebSphere® Application server using the W3C SOAP over JMS protocol. Configure the WebSphere MQ and WebSphere Application Server resources to develop and deploy Web service bound to W3C SOAP over JMS as a transport.

**Before you begin**

The task requires WebSphere Application Server v7.0.0.9 and WebSphere MQ v7.0.1.3.

**About this task**

The task has two steps:

**Procedure**

1. [Configure WebSphere MQ resources](#)
2. [Configure WebSphere Application Server resources](#)

**What to do next**

[Configure WebSphere MQ resources](#)

**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

**Related tasks**

[Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server](#)  
[Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)  
[Deploying to an Axis2 client using W3C SOAP over JMS](#)

**Configure WebSphere MQ resources**

**Before you begin**

For Axis2 support you required WebSphere MQ 7.0.1.3 or later.

**About this task**

For simplicity, the task assumes WebSphere MQ is installed on the same workstation as the other software, and uses bindings connections. The WebSphere Application Server and the Axis2 client configurations work with client connections. To run with task using client connections check that you can put and get messages to and from the request and reply queues from both the Axis2 client and the WebSphere Application Server computers.

Again, for simplicity, no security configuration is used. The user ID has full mqm authority.

**Procedure**

1. Create a default queue manager, QM1.

Use WebSphere MQ Explorer to create QM1 as a default queue manager. Configure it to start automatically and select the option to create a listener. Alternatively use the following commands:

```
crtmqm -q -sa QM1
strmqm
echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
         control(qmgr) replace | runmqsc
echo start listener(LISTENER.TCP) | runmqsc
```

2. Define a request queue, REQUESTAXIS, and a reply queue, REPLYAXIS. Use the Explorer, or the following commands:

```
echo define ql(REQUESTAXIS) replace | runmqsc
echo define ql(REPLYAXIS) replace | runmqsc
```

**What to do next**

[Configure WebSphere Application Server resources](#)

**Configure WebSphere Application Server resources****Before you begin**

For W3C SOAP over JMS support you require WebSphere Application Server v7. This configuration was performed on WebSphere Application Server Version 7.0 Test Environment v7.0.0.9 Update 1. WebSphere Application Server was supplied with Rational Software Architect for WebSphere Software 7.5.4. Rational Software Architect was updated to v7.5.5.1, by applying the latest updates that were available.

As part of the installation process, create a profile for WebSphere Application Server. In the task, administrative security is not enabled. The default profile name is was70profile1, and the server is server1.

**About this task**

Configure the WebSphere Application Server. You can either start the server from Rational Application Developer, and start the administrative console from the Servers view, or you can start the server using a command file and administer the server using a browser. The task uses the second method.

The server command files are in the folder, *Rational Installation Root\SDP\runtimes\base\_v7\profiles\was70profile1\bin*. The log files you might want to inspect are in *Rational Installation Root\SDP\runtimes\base\_v7\profiles\was70profile1\logs\server1*.

As a convention, all WebSphere MQ object names are uppercase, and all JNDI names referencing the WebSphere MQ objects are lowercase.

**Procedure**

1. Start the server.

```
startServer server1
```

2. Start a browser, open the administration console, and login.

```
http://localhost:9061/ibm/console/unsecureLogon.jsp
```

Type any string into the user ID field.

3. Create a connection factory, qm1
  - a. In the navigator, open **Resources > JMS > Connection factories**.
  - b. In the Connection factories window, select scope **Node=nodename**, click **New**.
  - c. Select **WebSphere MQ messaging provider > OK**.
  - d. Provide the queue manager connection information from [Table 1](#) > **Next**.

*Table 1. Queue manager connection information*

Field name	Value
Name	qm1
JNDI name	qm1

- e. Select **Enter all the required information into this wizard** as the connection method > **Next**.
- f. Type QM1 as the queue connection details > **Next**.
- g. Enter the connection details from [Table 2](#) > **Next**.

*Table 2. Connection details*

Field name	Value
Transport	Bindings, then client
Hostname	localhost
Port	1414
Server connection channel	SYSTEM.DEF.SVRCONN

- h. **Test connection > Next > Finish > Save**.



4. Create the JMS request queue, `requestaxis`.
  - a. In the Navigator, open **Resources > JMS > Queues**.
  - b. In the Connection factories window, select scope **Node=*nodename***, click **New**.
  - c. Select **WebSphere MQ messaging provider > OK**.
  - d. Enter the queue details from [Table 3](#) > **OK > Save**.

Table 3. Queue details

Field name	Value
Name	requestaxis
JNDI name	requestaxis
Queue name	REQUESTAXIS
Queue manager name	QM1

5. Repeat step 4 to create the JMS reply queue, `replyaxis`.
6. Create an activation specification, `qmlas`.  
The activation specification triggers the Web service router Message Driven Bean (MDB) when a message arrives on the request queue. The MDB is defined in the deployment descriptor of the Web service that is created by the Rational Application Developer Web service wizard.
  - a. In the Navigator, open **Resources > JMS > Activation specifications**.
  - b. In the Connection factories window, select scope **Node=*nodename***, click **New**.
  - c. Select **WebSphere MQ messaging provider > OK**.
  - d. Enter the basic attributes of the activation specification from [Table 4](#) > **Next**.

Table 4. Activation specification name

Field name	Value
Name	qmlas
JNDI name	qmlas

- e. Specify its MDB information from [Table 5](#) > **Next**.

Table 5. MDB information

Field name	Value
Destination JNDI name	requestaxis
Message selector	<i>Left blank</i>
Destination type	Queue

- f. Select **Enter all the required information into this wizard** as the connection method > **Next**.
- g. Type `QM1` as the queue connection details > **Next**.
- h. Enter the connection details from [Table 6](#) > **Next**.

Table 6. Connection details

Field name	Value
Transport	Bindings, then client
Hostname	localhost
Port	1414
Server connection channel	SYSTEM.DEF.SVRCONN

- i. **Test connection > Next > Finish > Save**.
7. Create a queue connection factory, `jms/WebServicesReplyQCF`, for the reply queue.  
The Web services router uses a queue connection factory to access a reply queue. In the deployment descriptor of the Web service the queue connection factory is given the default JNDI name of `jms/WebServicesReplyQCF`. You can change the name in the deployment descriptor. In this task, add the default name to the JMS resource definitions.
  - a. In the Navigator, open **Resources > JMS > Queue connection factories**.
  - b. In the Connection factories window, select scope **Node=*nodename***, click **New**.
  - c. Select **WebSphere MQ messaging provider > OK**.
  - d. Enter the basic attributes of the queue connection factory from [Table 7](#) > **Next**.

Table 7. Queue connection factory name

Field name	Value
Name	WebServicesReplyQCF
JNDI name	jms/WebServicesReplyQCF

- e. Select **Enter all the required information into this wizard** as the connection method > **Next**.
- f. Type `QM1` as the queue connection details > **Next**.
- g. Enter the connection details from [Table 6](#) > **Next**.


Table 8. Connection details

Field name	Value
Transport	Bindings, then client
Hostname	localhost
Port	1414
Server connection channel	SYSTEM.DEF.SVRCONN


- h. **Test connection > Next > Finish > Save**.

## What to do next

[Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)

 This build: January 26, 2011 11:10:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22552\_

## 1.6.6. Deploying a service to WebSphere ESB and Process Server service endpoint to use WebSphere Transport for SOAP

WebSphere® MQ transport for SOAP is not directly supported by WebSphere ESB and Process Server. You must configure a custom Export.

### About this task

WebSphere Integration Developer provides a SOAP data transformation that you can bind to the WebSphere MQ JMS Export to create a custom WebSphere MQ JMS SOAP Export.

Follow the instructions to create a customized Export to receive SOAP requests over WebSphere MQ transport for SOAP.


### Procedure

1. Read [Overview of imports and exports](#) and [How to connect to WebSphere MQ](#) in the WebSphere Process Server for Multiplatforms V6.2 Information Center.
2. Follow the task, [Generating an MQ JMS export binding](#) in the WebSphere Integration Developer, Version 6.2 Information Center. Use the SOAP data-binding described in [Prepackaged JMS data format transformations](#) to format the SOAP message.


**Parent topic:** [Deploying Web services using the WebSphere MQ transport for SOAP](#)

### Related information

[How to connect to WebSphere MQ](#)

 This build: January 26, 2011 11:09:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20450\_

## 1.7. Deploying Web service clients to use WebSphere MQ transport for SOAP

Deploy a Web service client to one of a number of different client environments and connect to a service using WebSphere® MQ transport for SOAP.

### Before you begin

Develop the Web service and deploy it to use WebSphere MQ transport for SOAP.

### About this task

You can deploy a web service client to run with WebSphere MQ transport for SOAP in a number of different client environments. You can deploy a Java client to Axis 1.4 using only the software installed with WebSphere MQ. For the other client environments, you must install additional software.

You are not restricted to running WebSphere transport for SOAP in the client environments for which there are deployment instructions. Use the instructions to deploy a client to one of the supported environments.

**Note:** Some integrated environments offer SOAP over JMS using the W3C recommended JMS SOAP binding, as well as the WebSphere MQ transport for SOAP binding. Releases of WebSphere MQ, up to and including 7.0.1.2, support only the WebSphere MQ transport for SOAP binding. From 7.0.1.3 onwards you can deploy Axis2 clients using a URI that conforms to the W3C candidate recommendation for SOAP over JMS. See the tutorial, [Develop a SOAP/JMS JAX-WS Web services application with WebSphere Application Server V7 and Rational Application Developer V7.5](#).

#### [Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)

Prepare a deployment directory and deployment descriptor for the client. Provide the client proxies and client class, and set up the CLASSPATH. Configure WebSphere MQ queues and channels, start the service and test the client.

#### [Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP](#)

Prepare a deployment directory and Axis2 configuration file for the client. Provide the client proxies and client class, and set up the CLASSPATH. Configure WebSphere MQ queues and channels, start the service and test the client.

#### [Deploying to an Axis2 client using W3C SOAP over JMS](#)

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 4 of connecting an Axis2 Web service client and a Web service deployed to WebSphere Application Server using the W3C SOAP over JMS protocol. Modify the URL in the Axis2 client developed for WebSphere MQ transport for SOAP to use the W3C candidate recommendation for SOAP over JMS.


#### [Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP](#)

Prepare a deployment directory and deployment descriptor for the client. Provide the client proxy and client class. Configure WebSphere MQ queues and channels, start the service and test the client.

**Parent topic:** [WebSphere MQ transport for SOAP](#)

 This build: January 26, 2011 11:09:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20500\_

## 1.7.1. Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP

Prepare a deployment directory and deployment descriptor for the client. Provide the client proxies and client class, and set up the `CLASSPATH`. Configure WebSphere® MQ queues and channels, start the service and test the client.

### Before you begin

**Tip:** Deploy the service to HTTP, develop and test the client for HTTP, and then modify the client for WebSphere MQ transport for SOAP:

1. Add the `Register.extension()` call to the client.
2. Change the static web service address in the client proxy locator class to use the URI for the WebSphere MQ transport for SOAP.

### About this task

Deploying an Axis 1.4 client to use WebSphere MQ transport for SOAP requires one additional deployment step compared to an HTTP client. You must create a client deployment descriptor, `client-config.wsdd`, to map the `jms:transport` to the sender class `com.ibm.mq.soap.transport.jms.WMQSender`.

If you use the command `amqwdeployWMQService` to generate client proxies, you can deploy the client using the directories the command generates.

### Procedure

1. Create a directory `deployDir` to hold the client deployment files.
2. Open a command window on Windows, or a command shell using X Window System on UNIX, in `deployDir`.
3. Run the `amqwsetcp.cmd` command to set the `CLASSPATH`.
4. Run the `amqwclientconfig.cmd` command to create an Axis 1.4 client deployment descriptor, `client-config.wsdd` in `deployDir`.
5. Make sure the classes in the client package, the client proxy classes, and the libraries the client uses, are in the `CLASSPATH`. `amqwdeployWMQService` places the .NET client proxies into `./generated/server/soap/client/remote/dotnetService` and the Axis 1.4 proxies into `./generated/server/soap/client/remote/client package`.

### Example

The example shows the configuration and output, [Figure 3](#), from an Axis 1.4 Java client. The client, [Figure 2](#), calls a Web service that echoes its input parameter. The service definition, [Figure 1](#), shows the URI taken from the service WSDL.

Figure 1. Service definition

```
<wsdl:service name="QuoteSOAPImplService">
  wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
    name="org.example.www.QuoteSOAPImpl_Wmq">
    <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
      &connectionFactory=(connectQueueManager(QM1)binding(server))
      &initialContextFactory=com.ibm.mq.jms.NoJndi
      &targetService=org.example.www.QuoteSOAPImpl.java
      &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
  </wsdl:service>
```

Figure 2. Axis 1.4 Java client

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
  public static void main(String[] args) {
    try {
      Register.extension();
      QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
      System.out.println("Response = "
        + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
    } catch (Exception e) {
      System.out.println("Exception = " + e.getMessage());
    }
  }
}
```

Figure 3. Client configuration and output

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

### What to do next

1. If you are deploying the client as a WebSphere MQ client, configure the client and server connection channel.
2. If you are deploying the client to a different queue manager to the service, you must make the destination queue available to the client. Configure the destination queue on the service queue manager as a cluster queue, or on the client queue manager as a remote queue definition.

**Parent topic:** [Deploying Web service clients to use WebSphere MQ transport for SOAP](#)

#### Related tasks


[Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP](#)  
[Deploying to an Axis2 client using W3C SOAP over JMS](#)  
[Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP](#)  
[Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using amqwdeployWMQService](#)

#### Related reference


[amqwdeployWMQService: deploy Web service utility](#)  
[amqwclientconfig: create Axis 1.4 Web services client deployment descriptor for WebSphere MQ transport for SOAP](#)  
[WebSphere MQ transport for SOAP sender](#)  
[Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender](#)

#### Related information

[Web services - Axis](#)

 This build: January 26, 2011 11:10:17

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts21510\_

## 1.7.2. Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP

Prepare a deployment directory and Axis2 configuration file for the client. Provide the client proxies and client class, and set up the CLASSPATH. Configure WebSphere® MQ queues and channels, start the service and test the client.

### Before you begin

**Tip:** Deploy the service to HTTP. Develop and test the client for HTTP, and then modify the URL to reference the service using WebSphere MQ transport for SOAP.

The task shows how to deploy an unmanaged Axis2 client to Java Standard Edition. You might want to deploy an Axis2 client to a Web container. In [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#), you developed a client in a Web container and deployed it to WebSphere Application Server Community Edition. As part of the server configuration, you enabled the Axis2 facet and included the facet in the configuration of the Web container. To configure Web containers on other application servers, refer to the Axis2 documentation, [http://ws.apache.org/axis2/1.4.1/installationguide.html#servlet\\_container](http://ws.apache.org/axis2/1.4.1/installationguide.html#servlet_container), or the documentation supplied with the Web server.

**Note:** Axis2 use the term, Servlet container. A Servlet container is the same as a Web container.

### About this task

Deploying an Axis2 client to use WebSphere MQ transport for SOAP is like deploying an Axis2 client to use HTTP. Additional steps are required to provide a classpath to the WebSphere MQ JAR files, and to modify the Axis2 configuration file. The Axis2 configuration file requires an additional entry for JMS. The entry refers to the WebSphere MQ transport for SOAP JAR file that implements the JMS `transportSender`.

Axis2 provides a script, `axis2.bat` or `axis2.sh`, which simplifies client deployment; see the examples in [Figure 4](#) and [Figure 5](#).

#### Note:

1. `axis2.bat` has a bug that must be corrected. The string `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` must be changed to `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`.
2. In `axis2.bat` and `axis2.sh`, `-Djava.ext.dirs` is used as a quick way to reference all the Axis2 JAR files, instead of adding them separately to the classpath. Unfortunately this approach is flawed, and only works with some JREs. It does not work with the IBM® JREs. The JVM parameter, `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`, makes the Axis JAR files available to the JVM. The JVM attempts to instantiate some of the Axis JAR files, and leads to an error, the details of which depend on the JVM. Typically, you might see one of the following lines in the stack trace: `org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)` OR `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`. The correct way to run an unmanaged Axis2 client is to add the Axis2 JAR files to the classpath. The classpath is available only to the client application and not to the JVM.

The procedure describes the general steps to run an unmanaged Axis2 client without using the `axis2` script. The examples in [Figure 2](#) and [Figure 3](#) are scripts for Windows and Linux.

### Procedure

1. Download Axis2 1.4.1 from <http://ws.apache.org/axis2/download/1.4.1/download.cqi> and unpack into a folder, `Axis2-1.4.1`.
2. Update `axis2.xml` in `Axis2-1.4.1\conf`.

- a. Update `axis2.xml` in `Axis2-1.4.1\conf`. Add WebSphere MQ transport for SOAP as a `transportSender`:

```
<transportSender name="jms"
class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
```

- b. If required, alter the size of the connection pool from the default of 10.

```
<transportSender name="jms"
class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
<parameter name="ResourcePoolCapacity">20</parameter>
</transportSender>
```

`ResourcePoolCapacity` defines how many service endpoint entries are kept in the cache. The value must be at least 1. If the number of service endpoint entries exceeds the cache size, entries are deleted to make room for new entries. The size of an endpoint entry varies.

Set a number that is large enough to avoid the cache thrashing.

See step 3 in [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#).

3. Create a directory *deployDir*. Under this directory copy the folder structure containing the client and client proxies. *deployDir* is equivalent to the *project\bin* folder in an Eclipse Java project.
4. Open a command window on Windows, or a command shell using X Window System on UNIX, in *deployDir*.
5. Update the classpath to include the current directory, Axis2 JAR files, *com.ibm.mqjms.jar* and *com.ibm.mq.axis2.jar*. *com.ibm.mqjms.jar* references all the other WebSphere MQ JAR files that are required.
6. Use the **Java** command to start the client program.

## Examples

Four example of running an Axis2 client are listed in [Figure 3](#) to [Figure 5](#). [Figure 1](#) shows the output from running the asynchronous client listed in [Figure 4](#).

Figure 1. Output from running SQA2AsyncClient

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

Figure 2. runpojo.bat: Windows, using a classpath

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
" .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

Figure 3. runpojo.sh: Linux, using a classpath.

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
    AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME": "$JAVA_HOME/lib/tools.jar": "$AXIS2_CLASSPATH": "$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

Figure 4. runaxis2.bat: Windows, using axis2.bat [Note](#)

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

Figure 5. runaxis2.sh: Linux, using axis2.sh [Note](#)

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

**Parent topic:** [Deploying Web service clients to use WebSphere MQ transport for SOAP](#)

## Related tasks

[Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)


[Deploying to an Axis2 client using W3C SOAP over JMS](#)

[Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP](#)

[Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using amqwdployWMQService](#)

## Related information

[Welcome to Apache Axis2/Java](#)

 This build: January 26, 2011 11:10:17

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts21512\_

### 1.7.3. Deploying to an Axis2 client using W3C SOAP over JMS

A Web service bound to the W3C candidate recommendation for SOAP over JMS must run in the EJB container of a JEE application server. This task is step 4 of connecting an Axis2 Web service client and a Web service deployed to WebSphere® Application Server using the W3C SOAP over JMS protocol. Modify the URL in the Axis2 client developed for WebSphere MQ transport for SOAP to use the W3C candidate recommendation for SOAP over JMS.

#### Before you begin

You must first complete the task, [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#) to call `SimpleJavaListener` using an Axis2 client and the WebSphere MQ transport for SOAP protocol.

You must also have created the Web service and configured WebSphere MQ and WebSphere Application Server in the previous tasks:

1. [Configure WebSphere MQ resources.](#)
2. [Configure WebSphere Application Server resources.](#)
3. [Developing a JAX-WS EJB Web service for W3C SOAP over JMS.](#)

In the task, the client runs in Eclipse Galileo. You might run the client from the command line by modifying the `Axis2.bat` file shipped with Axis2.

#### About this task

The only change you must make to the existing `Axis2 StockQuoteAxis` static client to call the `StockQuoteAxis` service hosted by WebSphere Application Server is to change the URL passed to the client. Since the WSDL has not changed, you can use the same proxy classes in the `soap.server` package.

You have two approaches to defining the URL to pass to the client. You might use the same URL as in the generated `StockQuoteAxis.wsdl`. You must add the `jndiInitialContextFactory` and the `jndiURL` parameters to access the WebSphere Application Server JNDI directory. Another approach is to change the URL and give the client direct access to the `REQUESTAXIS` and `REPLYAXIS` queues on `QM1`, without using a JNDI lookup.

The connection parameters you define in the URL passed to the Axis2 client are used to connect to the WebSphere MQ queue manager and queues required to send and receive SOAP messages. The connection parameters passed to the Axis2 client are not necessarily used by the service. You can use the distributed queuing capabilities of WebSphere MQ to decouple the client and service from using the same queue manager, or the same name server.

#### Procedure

1. Save the URL from the generated `StockQuoteAxis.wsdl` and close down Rational® Application Developer to save on memory.

If you did not change the server configuration, closing Rational Application Developer stops the application server. In which case start the server with the command:

```
startserver server1
```

2. Open Eclipse Galileo in the workspace with the Axis2 client project.
3. Open `SQA2StaticClient.java`.  
See [SQA2StaticClient.java](#).
4. Call the service using the `queue` variant of the URI.

- a. Modify the URL. The new URI is:

```
jms:queue:REQUESTAXIS
    ?replyToName=REPLYAXIS
    &connectionFactory=connectQueueManager(QM1)Bind(Server)
    &targetService=StockQuoteAxis;
```

Compare this to the URL from `StockQuoteAxis.wsdl`:

Figure 1. URL from `StockQuoteAxis.wsdl`

```
jms:jndi:requestaxis
    ?jndiConnectionFactoryName=qm1
    &targetService=StockQuoteAxis
```

- `REQUESTAXIS` is now uppercase as it is a queue name and not a JNDI name.
- The connection to `QM1` is straightforward.
- The URI does not contain the name of the Reply to destination. The client must define the queue it expects replies on.

- b. Run `SQA2StaticClient.java` using the same Run as... configuration as you did in the task, [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#).

5. Call the service using the `jndi` variant of the URI, using WebSphere Application Server as the naming server.

- a. Use the URL from `StockQuoteAxis.wsdl`, [Figure 1](#), providing the missing parameters to use the naming service in WebSphere Application Server.

The missing parameters and values you must provide are:

Table 1. Additional JNDI parameters

Parameter	Value used in this example	Description
<code>&amp;jndiURL</code>	<code>iiop://localhost:2810</code> or <code>corbaname:iiop:localhost:2810</code>	URI of naming provider. For WebSphere Application Server the value defaults to 2809. It is also known as the port number of the RMI connector, and the bootstrap port. The value is listed in the <code>SystemOut.log</code> <code>00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810</code>
<code>&amp;jndiInitialContextFactory</code>	<code>com.ibm.websphere.naming.WsnInitialContextFactory</code>	The name of the initial context factory used by WebSphere Application Server.
<code>&amp;replyToName</code>	<code>replyaxis</code>	JNDI name of <code>REPLYAXIS</code> queue.

```
jms:jndi:requestaxis?
    &jndiURL=iiop://localhost:2810
    &jndiConnectionFactoryName=qm1
    &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
    &targetService=StockQuoteAxis
    &replyToName=replyaxis;
```

- b. Add the JAR files required by the JNDI lookup.

In this configuration, the following JAR files were added to the build path to run the task using the `jndi` variant of the JMS url:

- `com.ibm.jaxws.thinclient_7.0.0.jar` from *Rational install directory\SDP\runtimes\base\_v7\runtimes.*
- `com.ibm.ws.runtime.jar` from *Rational install directory\SDP\runtimes\base\_v7\plugins*

For a different JNDI provider you require different JAR files.

The other JAR files in the build path are:

- i. All the JAR files in *WebSphere MQ Install directory\java\lib.*
  - ii. All the JAR files in *Axis2-1.5.1\lib.*
  - iii. Java 1.5 JRE.
- c. Run `SQA2StaticClient.java` using the same Run as... configuration as you did in the task, [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#).


## Results

In both cases the reply from the service is displayed in the client console view.

**Parent topic:** [Deploying Web service clients to use WebSphere MQ transport for SOAP](#)

## Related tasks

[Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)  
[Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP](#)  
[Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP](#)  
[Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server](#)  
[Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)  
[Configuring WebSphere Application Server to use W3C SOAP over JMS](#)  
[Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22553\_

## 1.7.4. Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP

Prepare a deployment directory and deployment descriptor for the client. Provide the client proxy and client class. Configure WebSphere® MQ queues and channels, start the service and test the client.

### Before you begin

**Tip:** Develop and test the service and the client using Visual Studio. Then modify the client for WebSphere MQ transport for SOAP.

1. If you are deploying a service using .NET Framework 1 or 2, build the service as a library (.dll). Deploy using WebSphere MQ transport for SOAP.
2. Add the `Register.Extension()` call to the client.
3. Add a reference to `amqsoap.dll`, which is located in `MQ_Install\bin`.
4. Change the static `Url` property in the client proxy class constructor to the `jms:/ URI`, for WebSphere MQ transport for SOAP.

### About this task

Deploying a Web service client for .NET Framework 1 or 2 to use WebSphere MQ transport for SOAP requires an additional deployment step. You need to register `amqsoap.dll` with the .NET Framework. `amqsoap.dll` is automatically registered as part of installing WebSphere MQ transport for SOAP, but you might need to register it again.

If you use the command `amqwdeployWMQService` to generate client proxies, you can deploy the client using the directories the command generates.

### Procedure

1. Create a directory `deployDir` to hold the client deployment files.
2. Open a command window in `deployDir`.
3. Run `amqwsetcp` to set the `CLASSPATH` if the service is to run on Axis 1.4.
4. If necessary, run `amqwRegisterDotNet` to register `amqsoap.dll` with the .NET Framework.

### Example

The example shows the configuration and output, [Figure 3](#), from an .NET Framework V2 client. The client, [Figure 2](#), calls a Web service that echoes its input parameter. The static `Url` definition, [Figure 1](#), shows the constructor for the client proxy.

Figure 1. Static client proxy constructor

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
    &connectionFactory=(connectQueueManager(QM1)binding(server))
    &initialContextFactory=com.ibm.mq.jms.NoJndi
    &targetService=Quote.asmx
    &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

Figure 2. Client program

```

using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
}

```

Figure 3. Configuration and output

```

C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM

```

### What to do next

1. If you are deploying the client as a WebSphere MQ client, configure the client and server connection channel.
2. If you are deploying the client to a different queue manager to the service, you must make the destination queue available to the client. Configure the destination queue on the service queue manager as a cluster queue, or on the client queue manager as a remote queue definition.


**Parent topic:** [Deploying Web service clients to use WebSphere MQ transport for SOAP](#)

### Related tasks

[Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)  
[Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP](#)  
[Deploying to an Axis2 client using W3C SOAP over JMS](#)

### Related reference

[amqwdeployWMQService: deploy Web service utility](#)  
[amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET](#)  
[WebSphere MQ transport for SOAP sender](#)  
[Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender](#)

 This build: January 26, 2011 11:10:18

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts21520\_

## 1.8. Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server

When you complete this task you will have called a JAX-WS Web service running in WebSphere® Application Server from an Axis2 client. The Axis2 client and WebSphere Application Server use the W3C candidate recommendation for the SOAP over JMS protocol running on WebSphere MQ. Use Eclipse Galileo and Rational® Application Developer to build the Web service client and Web service, respectively.

### Before you begin

The task requires version 7 of Rational Software Development Environment and WebSphere Application Server. The task was created using the Rational Application Developer packaged with Rational Software Architect for WebSphere Software v7.5.5.1, and WebSphere Application Server Version 7.0 Test Environment v7.0.0.9 Update 1. You also require WebSphere MQ v7.0.1.3.

The task builds on two other tasks, [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#), and [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#). To complete these tasks your development environment already has Eclipse Galileo, WASCE, the Eclipse plugin for WASCE, and Axis2 1.4.1 installed. You do not require WASCE for this task.

Some of the steps are complex. The steps assume some familiarity with developing Web service applications for WebSphere Application Server using Rational Application Developer. The processor and memory demands of the task are large. The task was performed in a VMWare Windows XP SP3 virtual machine allocated 1.8GB of memory.

Install all the software before starting the task. The software takes about a day to download and a day to install, depending on bandwidth. The task takes at least half a day.

### About this task

The scenario for this task is that you have developed a stock quotation Web service, `StockQuoteAxis`, using an open source tool, Eclipse Galileo. `StockQuoteAxis` is deployed using SOAP over HTTP running on an open source server, WASCE.

You want to bind the Web services you deploy to a standards-based messaging transport, such as SOAP over JMS, or to Web services reliable messaging, as well as to SOAP over HTTP. You want both the client and service to use standards-based interfaces. For this reason, although your future projects development team have implemented a solution using WebSphere MQ transport for SOAP, you have not gone into production.

The Axis2 client has removed the problem that the SOAP client for the WebSphere MQ transport for SOAP required a change from the HTTP client. The problem still remained that the service connected by the WebSphere MQ transport for SOAP is hosted by a special listener provided by WebSphere MQ: `SimpleJavaListener`.

With the W3C SOAP over JMS standard in candidate recommendation status, some vendors are providing support for W3C SOAP over JMS. The support enables you to deploy a Web service to an application server and connect to the same service using a variety of connectivity protocols. The support provided by WebSphere Application Server v7 removes problem of having to host the Web service separately in order to use a message-based SOAP transport. The use of a standards-based message transport interface, JMS, means you can develop solutions using tools from different vendors. You hope the Web services tools in Eclipse will include SOAP over JMS bindings in the future.

Most of the steps are performed using Eclipse, or the management tools provided with the WebSphere products. The steps are described for a




Windows environment. With slight modifications to some commands, you can perform the steps on other platforms.

The preliminary steps creating the HTTP Web service, and connecting to it using Axis2 are listed. The client, and WSDL, from these steps are used to create the solution

## Procedure

1. Connect to the StockQuoteAxis Web service using an Axis2 client and WebSphere MQ transport for SOAP
  - a. [Developing a JAX-RPC service for WebSphere MQ transport for SOAP using Eclipse](#)
  - b. [Developing a JAX-WS client for WebSphere transport for SOAP using Eclipse](#)
  - c. [Deploying a Web service client to Axis2 to use WebSphere MQ transport for SOAP](#)
2. Connect to the StockQuoteAxis Web service using an Axis2 client and the W3C candidate recommendation for SOAP over JMS.
  - a. [Configure WebSphere MQ resources](#)
  - b. [Configure WebSphere Application Server resources](#)
  - c. [Developing a JAX-WS EJB Web service for W3C SOAP over JMS](#)
  - d. [Deploying to an Axis2 client using W3C SOAP over JMS](#)

**Parent topic:** [WebSphere MQ transport for SOAP](#)

 This build: January 26, 2011 11:10:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22550\_

## 1.9. Reference

WebSphere MQ transport for SOAP reference information arranged alphabetically.

### [amqwSOAPNETListener: WebSphere MQ SOAP listener for .NET Framework 1 or 2](#)

Syntax and parameters for the WebSphere MQ SOAP listener for .NET Framework 1 or 2.

### [amqswsdl: generate WSDL for .NET Framework 1 or 2 service](#)

**amqswsdl** takes a Web service written for .NET Framework 1 or 2, and generates the WSDL for the class, inserting the URI you provide for the WebSphere MQ transport for SOAP into the generated WSDL.

### [amqwclientconfig: create Axis 1.4 Web services client deployment descriptor for WebSphere MQ transport for SOAP](#)

**amqwclientconfig** creates the `client-config.wsdd` Axis 1.4 client deployment descriptor file.

### [amqwdeployWMOService: deploy Web service utility](#)

The deployment utility prepares a service class for use as a Web service using WebSphere MQ as the transport.

### [amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET](#)

Register WebSphere MQ transport for SOAP to the global assembly cache on .NET.

### [Apache software license](#)

Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>

### [MQMD SOAP settings](#)

The WebSphere MQ SOAP sender and WebSphere MQ SOAP listener create a message descriptor (**MQMD**). The topic describes the fields you must set in the MQMD if you create your own SOAP sender or listener.

### [MORFH2 SOAP settings](#)

The WebSphere MQ SOAP senders and listeners create an MORFH2 with the following settings to send a SOAP message.

### [runivt: WebSphere MQ transport for SOAP installation verification test](#)

An installation verification test suite (IVT) is provided with WebSphere MQ transport for SOAP. **runivt** runs a number of demonstration applications and ensures that the environment is correctly set up after installation.

### [Secure Web services over WebSphere MQ transport for SOAP](#)

You might secure Web services that use the WebSphere MQ transport for SOAP in one of two ways. Either create an SSL channel between the client and server, or use Web services security.

### [SimpleJavaListener: WebSphere MQ SOAP Listener for Axis 1.4](#)

Syntax and parameters for the WebSphere MQ SOAP listener for Axis 1.4.

### [WebSphere MQ SOAP listeners](#)

A WebSphere MQ SOAP listener reads an incoming SOAP request from the queue specified as the destination in the URI. It checks the format of the request message and then invokes a Web service using the Web services infrastructure. A WebSphere MQ SOAP listener returns any response or error from a Web service using the reply destination queue in the URI. It returns WebSphere MQ reports to the reply queue.

### [WebSphere MQ transport for SOAP sender](#)

Sender classes are provided for Axis and .NET Framework 1 and .NET Framework 2. The sender constructs a SOAP request and puts it on a queue, it then blocks until it has read a response from the response queue. You can alter the behavior of the classes by passing different URIs from a SOAP client. For .NET Framework 3 use WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF).

### [Transactions](#)

Use the `-x` option when starting the listener, to run Web services transactionally. Select the integrity of messages by setting the `persistence` option in the service URI.

### [URI syntax and parameters for Web service deployment](#)

The syntax and parameters to deploy a WebSphere MQ Web service are defined in a URI. The deployment utility generates a default URI based on the name of the Web service. You can override the defaults by defining your own URI as a parameter to the deployment utility. The



The deployment utility checks for the compatibility of the `-x` and `-a` flags. If `-x none` is specified, then `-a LowMsgIntegrity` must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

#### **-d msec**

`msecs` specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If `msecs` is set to `-1`, the listener stays alive indefinitely.

#### **-i Context**

`Context` specifies whether the listeners pass identity context. `Context` takes the following values:

##### **passContext**

Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the `MQOPEN` call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed `MQOPEN`. The listener then continues to process subsequent incoming messages as normal.

##### **ownContext**

The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

#### **-n numThreads**

`numThreads` specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

#### **-v**

`-v` sets verbose output from external commands. Error messages are always displayed. Use `-v` to output commands that you can tailor to create customized deployment scripts.

#### **-w serviceDirectory**

`serviceDirectory` is the directory containing the web service.

#### **-x transactionality**

`transactionality` specifies the type of transactional control for the listener. `transactionality` can be set to one of the following values:

##### **onePhase**

WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

##### **twoPhase**

Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

##### **none**

No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the `-x` and `-a` flags. See the description of the `-a` flag for details.


#### **.NET Example**

```
amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.Nojndi"
-w C:/wmqsoap/demos
-n 20
```

Parent topic: [Reference](#)

#### **Related reference**

[WebSphere MQ SOAP listeners](#)

 This build: January 26, 2011 11:09:37

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20130\_

## 1.9.2. amqswsdl: generate WSDL for .NET Framework 1 or 2 service

`amqswsdl` takes a Web service written for .NET Framework 1 or 2, and generates the WSDL for the class, inserting the URI you provide for the WebSphere® MQ transport for SOAP into the generated WSDL.

#### **Purpose**

Use `amqswsdl` to generate WSDL containing the URI of the service deployed to WebSphere MQ. Use the WSDL to generate client proxies.

```
>>-amqswsdl-- -escapedUri-- -className-- .asmx-- -className-- .wsdl-><
```

#### **Parameters**

##### **escapedUri (Input)**

The URI of the service, with all "&" escaped to "&amp;". For example:

```
"jms:/queue?destination=REQUESTDOTNET
&amp;.initialContextFactory=com.ibm.mq.jms.Nojndi
&amp;.connectionFactory=(connectQueueManager(QM1)binding(server))
&amp;.targetService=Quote.asmx"
```

##### **className.asmx (Input)**

The service class.

**className.wsdl (Output)**

The service WSDL.

**Description**

If the class is implemented using the code-behind programming model, you must build `className.dll` and store it in `./bin`.


**Parent topic:** [Reference](#)

**Related tasks**

[Developing Web services for WebSphere MQ transport for SOAP](#)

**Related reference**

[Supported Web services](#)

 This build: January 26, 2011 11:09:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20480\_

### 1.9.3. amqwclientconfig: create Axis 1.4 Web services client deployment descriptor for WebSphere MQ transport for SOAP

**amqwclientconfig** creates the `client-config.wsdd` Axis 1.4 client deployment descriptor file.

**Purpose**

It adds the `jms:/` transport to the descriptor, and registers `java.com.ibm.mq.soap.transport.jms.WMQSender` as the class to handle SOAP requests for the `jms: transport`.

Syntax

```
>>-amqwclientconfig-----<<
```

**Description**

**amqwclientconfig** calls **amqwsetcp** to set the `CLASSPATH` and runs the command:

```
java org.apache.axis.utils.Admin client "%WMO SOAP_HOME%\bin\amqwclientTransport.wsdd"
```


**Parent topic:** [Reference](#)

**Related tasks**

[Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)

**Related reference**

[WebSphere MQ transport for SOAP Web service clients](#)

 This build: January 26, 2011 11:09:36

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20110\_

### 1.9.4. amqwdeployWMQService: deploy Web service utility

The deployment utility prepares a service class for use as a Web service using WebSphere® MQ as the transport.

**Purpose**

Use the deployment utility to generate the files that are needed to deploy an Axis 1.4, .NET Framework 1 or .NET Framework 2 service. Use the files to deploy a service invoked by WebSphere MQ. The files generated by **amqwdeployWMQService** are shown in [Output files from amqwdeployWMQService](#).

Syntax diagram

```
>>-----<<
```

UNIX

```
|--./amqwdeployWMQService.sh--+-f -className+----->
|_ -? -----'
```

```
>>+-----+--| Optional parameters |-----|
| .allAxis-----|
|'- -c +-compileJava-----+
| +genAxisWsd-----+
| +axisDeploy-----+
| +genProxiesToAxis---+
| +genProxiesToDotNet-+
| +genAxisWMOBits-----+
|'-startWMOmonitor----'
```

Windows

```
|--amqwdeployWMQService--+-f -className+----->
```

```

'- -? -----'

>-----| Optional parameters |-----|
|      .-allAsmx-----. |
| '- -c --compileJava-----+'
|      +-genAxisWsd-----+
|      +-genAsmxWsd-----+
|      +-genProxiestoAxis--+
|      +-genProxiesToDotNet--+
|      +-genAsmxWmqBits-----+
|      '-startWmqMonitor----+'

Optional parameters

|----->
|      .-DefaultMsgIntegrity--+-----+
|      |                          |      .-onePhase-. | | |
|      |                          |      '- -x --twophase--+' | | |
| '- -a --HighMsgIntegrity--+-----+'
|      |                          |      .-onePhase-. | | |
|      |                          |      '- -x --twophase--+' | | |
|      '-LowMsgIntegrity--+-----+'
|      |                          |      .-onePhase-. | | |
|      |                          |      '- -x --twophase--+' | | |
|      |                          |      '-none-----' | | |

>----->
|      .-3-----. | |      .-passContext-. |
| '- -b --bothresh--+ ' '- -i --ownContext----+'

>----->
|      .-10-----. | | '- -r ' ' -s '
| '- -n --numThreads--+ '

>----->
|      .-runmqtrm-----. | '- -tmq -initQueueName-'
| '- -tmp --programName--+ '

>----->
+| URI (.NET) |++ '- -v ' |      .-rpcEncoded-. |
| '-| URI (Java) |- '      '- -w --rpcLiteral++-'

|-----|
|      .-onePhase-----. |
| '- -x --twophase-----+'
|      '-none-- -a -LowMsgIntegrity-'

```

## Required parameters

### -f *className*

*className* is the name of the class to be deployed. For Axis services *className* is the Java source file, and for .NET services, the `.asmx` file. [Figure 1](#) illustrates the deployment of an Axis service and [Figure 2](#) of a .NET service.

Figure 1. Example deployment of Axis service

```
amqwdeployWmqService -f javaDemos/service/StockQuoteAxis.java
```

Figure 2. Example deployment of .NET service

```
amqwdeployWmqService -f StockQuoteDotNet.asmx
```

For Java, *className* must be fully qualified by the package name. It can be specified as a path name with directory separators or as a class name with period separators. The generated class is located in `./generated/client/remote/path name`. For a .NET service, although the directory can be specified, generated Java proxies are always located in `./generated/client/remote/dotNetService`.

If you specify a URI with the `-u` option and within the URI specify *targetService*, the deployment utility checks the *className*. *className* must match *targetService*. If the class and service do not match, the deployment utility displays an error message and exits.

### -?

Print out help text describing how the command is used.

## Optional parameters

### -a *integrityOption*

*integrityOption* specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

#### DefaultMsgIntegrity

For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the `-a` option is omitted, or if *integrityOption* is not specified.

#### LowMsgIntegrity

For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

#### HighMsgIntegrity

For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the `-x` and `-a` flags. If `-x none` is specified, then `-a LowMsgIntegrity` must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

### -b *bothresh*

*bothresh* specifies the backout threshold setting for the request queue. The default is 3.

### -c *operation*

*operation* specifies which part of the deployment process to execute. *operation* is one of the following options:

**allAxis**

Perform all compile and setup steps for an Axis or Java service<sup>1</sup>.

**compileJava**

Compile the Java service: `.java` to `.class`.

**genAxisWSDL**

Generate WSDL: `.class` to `.wsdl`.

**axisDeploy**

Deploy the class file: `.wsdl` to `.wsdd`, apply `.wsdd`.

**genProxiesToAxis**

Generate proxies: `.wsdl` to `.java` and `.class`.

**genAxisWMQBits**

Set up WebSphere MQ queues, WebSphere MQ SOAP listeners and triggers for an Axis service.

**allAsmx**

Perform all setup steps for a .NET service<sup>2</sup>.

**genAsmxWSDL**

Generate WSDL: `.asmx` to `.wsdl`.

**genProxiesToDotNet**

Generate proxies: `.wsdl` to `.java`, `.class`, `.cs` and `.vb`.

**genAsmxWMQBits**

Set up WebSphere MQ queues, WebSphere MQ SOAP listeners and triggers

**startWMQMonitor**

Start the trigger monitor for WebSphere MQ SOAP services.

**Note:** `runmqtrm` runs under the `mqm` user ID. If security is an issue, you must ensure that the listeners are started under appropriate user IDs.

**-i Context**

*Context* specifies whether the listeners pass identity context. *Context* takes the following values:

**passContext**

Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the `MQOPEN` call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed `MQOPEN`. The listener then continues to process subsequent incoming messages as normal.

**ownContext**

The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

**-n numThreads**

*numThreads* specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

**-r**

`-r` specifies that any existing request or trigger monitor queue definitions are replaced. Trigger monitor queues are replaced only if `-tmq` is also specified. Queues are re-created with standard default attributes and existing messages on the queues are deleted. If the `-r` option is not used then any existing queue definitions are not altered and existing messages are not deleted. By not specifying `-r`, you ensure that any customized queue attributes are preserved.

**-s**

Configure the listener to run as a WebSphere MQ service. If `-s` and `-tmq` are both specified, the deployment utility displays an error message and exits.

**-tmp programName**

*programName* specifies the name of a trigger monitor program. Use `-tmp programName` in a UNIX environment as an alternative to using `runmqtrm`. Programs it initiates run under `mqm` authority.

For example:

```
amqwdeployWMQService -f javaDemos/service/StockQuoteAxis.java
-tmq trigger.monitor.queue -tmp trigmon
```

**-tmq queueName**

*queueName* specifies a trigger monitor queue name. WebSphere MQ process definitions are created to configure automatic triggering of WebSphere MQ SOAP listeners with the associated trigger monitor queue name. If the option is not specified then no triggering configuration is defined by the deployment utility. If `-s` and `-tmq` are both specified, the deployment utility displays an error message and exits.

**URI platform**

See [URI syntax and parameters for Web service deployment](#).

**-v**

`-v` sets verbose output from external commands. Error messages are always displayed. Use `-v` to output commands that you can tailor to create customized deployment scripts.

**➤-w◀**

➤-w controls the style of WSDL to generate. The default is `rpcEncoded`, for compatibility with previous releases of WebSphere MQ transport for SOAP. Use `rpLiteral` to create WSDL compatible with Axis2 client proxy generation. `rpcEncoded` is not compatible with WS-I recommendations.◀

**-x transactionality**

*transactionality* specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

**onePhase**

WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

**twoPhase**

Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

#### none

No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

#### Errors

On Windows, if errors are reported from **amqswsdl**, try issuing the following command to register **.asmx** files as services.

```
%windir%/Microsoft.NET/Framework/version number/aspnet_regiis.exe -ir
```

The problem typically occurs on systems where IIS has not been installed, or IIS has been installed after .NET. The problem is encountered when **amqswsdl** generates the **.wsdl** files.

**Note:** The registry keys are also required to permit the listener to invoke the services. If you use your own customized deployment procedures, you might not encounter the problem until run time.

#### [Output files from amqwdeployWMQService](#)

A list of the directories and files output from **amqwdeployWMQService**

#### [Usage notes for amqwdeployWMQService](#)

Describes the tasks performed by **amqwdeployWMQService**.

**Parent topic:** [Reference](#)

#### Related tasks

[Deploying Web services using the WebSphere MQ transport for SOAP](#)

#### Related reference

[URI syntax and parameters for Web service deployment](#)


[Secure Web services over WebSphere MQ transport for SOAP](#)

#### Related information

[System requirements for WebSphere MQ](#)

<sup>1</sup> Default if *className* has a **.java** extension

<sup>2</sup> Default if *className* has a **.asmx** extension.

 This build: January 26, 2011 11:09:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20730\_

### 1.9.4.1. Output files from amqwdeployWMQService

A list of the directories and files output from **amqwdeployWMQService**

Table 1. Output files from **amqwdeployWMQService**

Outputs	Description	Output directory	Filename
<b>.class</b>	Compiled Java source file	<i>./generated/server/server package</i>	<i>classname.class</i>
<b>.wsdl</b>	Service description	<i>./generated</i>	<i>classNameAxis_Wmq.wsdl</i> <i>classNameDotNet_Wmq.wsdl</i>
<b>.wsdd</b>	Axis client and service deployment files	<i>./</i>	<i>client-config.wsdd</i> <i>server-config.wsdd</i>
		<i>./generated/server/server package</i>	<i>className_deploy.wsdd</i> <i>className_undeploy.wsdd</i>
Client source (.vb, .cs, .java)	.Net client stubs to Axis service	<i>./generated/client</i>	<i>classNameAxisService.cs</i> <i>classNameAxisService.vb</i>
	.Net client stubs to .Net service	<i>./generated/client</i>	<i>classNameDotNet.cs</i> <i>classNameDotNet.vb</i>
Client helper (.java and .class)	Java client proxies to .Net service	<i>./generated/server/soap/client/remote/dotnetService</i>	<i>classNameDotNet.class</i> <i>classNameDotNet.java</i> <i>classNameDotNetLocator.class</i> <i>classNameDotNetLocator.java</i> <i>classNameDotNetSoap12Stub.class</i> <i>classNameDotNetSoap12Stub.java</i> <i>classNameDotNetSoap_BindingStub.class</i> <i>classNameDotNetSoap_BindingStub.java</i> <i>classNameDotNetSoap_PortType.class</i> <i>classNameDotNetSoap_PortType.java</i>
	Java client proxies to Axis service	<i>./generated/server/soap/client/remote/client package</i>	<i>SoapServerclassNameAxisBindingSoapStub.class</i> <i>SoapServerclassNameAxisBindingSoapStub.java</i> <i>classNameAxis.class</i> <i>classNameAxis.java</i> <i>classNameAxisService.class</i> <i>classNameAxisService.java</i> <i>classNameAxisServiceLocator.class</i> <i>classNameAxisServiceLocator.java</i>

Scripts (.cmd and .sh)	Listener scripts	/generated/server	startWMQJListener.cmd startWMQJListener.sh startWMQNListener.cmd endWMQJListener.cmd endWMQJListener.sh endWMQNListener.cmd
------------------------------	---------------------	-------------------	--

Parent topic: [amqwdeployWMQService: deploy Web service utility](#)

This build: January 26, 2011 11:09:58

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20732\_

## 1.9.4.2. Usage notes for amqwdeployWMQService

Describes the tasks performed by **amqwdeployWMQService**.

The deployment utility performs the following actions.

- Checks paths to the following files:
  - `axis.jar`.
  - `WMQSOAP_HOME/java/lib/com.ibm.mq.soap.jar`.
  - On Windows, `csc.exe`
- On Windows, uses either `%SystemRoot%\Microsoft.NET\Framework\v1.1.432` or, if the C# compiler is installed, the path to `csc.exe` as the path to the .NET Framework.
 

**Note:** If you have Microsoft Visual Studio 2008 installed (Version 9), `wsdl.exe` is not in the path to `csc.exe`. You need to add the path to the .NET framework to your Path variable; for example:

```
Set Path=C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;%Path%
```
- Creates the `./generated` directory, and required subdirectories, if they do not exist.
- For Java services, compiles the source into `className.class`.
- Generates WSDL.
- For Java services, creates deployment descriptor files `className_deploy.wsdd` and `className_undeploy.wsdd`
- For Java services, creates or updates the Axis deployment descriptor file, `server-config.wsdd`.
- Generates the client proxies for Java, C# and Visual Basic from the WSDL.
 

**Note:** On Windows, the deploy utility generates proxies for Visual Basic and C# regardless of the language in which the service is written. The WSDL and the proxies generated from it include the appropriate URI to call the service:

  - Figure 1. Example URI in generated .NET client to call .NET service*

```
jms:/queue?destination=SOAPN.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.NoJndi
&targetService=StockQuoteDotNet.asmx
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```
  - Figure 2. Example URI in generated .NET client to call Axis 1 service*

```
jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.NoJndi
&targetService=soap.server.StockQuoteAxis.java
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```
- Compiles the Java proxies.
- Creates a WebSphere® MQ queue, `requestQueue` to hold requests for the service. The default queue name is of the form `SOAPJ.directory`, or you can specify `requestQueue` in the `-u` URI option.
- Creates command and shell script files to start the WebSphere MQ SOAP listeners that process the request queue.
- If the `-tmq` option has been used, the deployment utility creates WebSphere MQ definitions to trigger WebSphere MQ SOAP listener processes automatically.
  - The deployment utility uses the `APPLICID` attribute of the `runmqsc DEFINE PROCESS` command to contain a command to start the listener. The command has the name of the deployment directory embedded in it. The `APPLICID` field has a maximum length of 256, which limits the maximum length of the deployment directory. The directory limit for Java services is as follows:
    - UNIX: 218
    - Windows: 197 minus the length of the request queue name.
  - For .NET services, the directory limit is as follows:
    - Windows: 209 minus length of the service name, minus the `.asmx` extension.
  - The deployment utility checks whether the limit for `APPLICID` is exceeded. If the limit is exceeded, the utility does not attempt to define the triggering process. It displays an error message and the deployment process fails without performing any deployment steps.

The following examples show the configuration and start commands generated by the deployment utility to start a WebSphere MQ SOAP listener.

Figure 3. WebSphere MQ configuration commands to trigger a SOAP listener.

```
DEFINE PROCESS(requestQueue) APPLICID(applicIDStr) REPLACE
ALTER QLOCAL(requestQueue) TRIGTYPE(FIRST) TRIGGER
PROCESS(requestQueue) INITQ(initQueueName) TRIGMPRI(0)
```

Figure 4. Starting Axis SOAP listener on Windows

```
applicIDStr = start "Java WMQSoapListener -requestQueue"
/min .\generated\server\startWMQJListener.cmd;
```



Figure 5. Starting .NET SOAP listener on Windows

```
applicIDStr = start "WMQAsmxListener -className\  
/min .\generated\server\startWMQNListener.cmd;
```

Figure 6. Starting Axis SOAP listener on UNIX

```
applicIDStr = xterm -iconic -T \"Java WMQSoapListener_requestQueue\  
-e ./generated/server/startWMQJListener.sh & #
```

Parent topic: [amqwdeployWMQService: deploy Web service utility](#)

This build: January 26, 2011 11:09:59

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20734\_

## 1.9.5. amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET

Register WebSphere® MQ transport for SOAP to the global assembly cache on .NET.

### Purpose

**amqwRegisterdotNet** registers the WebSphere MQ SOAP sender, SOAP listener, and WSDL processor with .NET Framework 1 or 2.

### Syntax

```
>>-amqwRegisterdotNet-----<<
```

### Description

**amqwRegisterdotNet** is run automatically during installation. You do not need to run it again if the .NET Framework you are using was installed before WebSphere MQ transport for SOAP. You can run it as many times as you want. Use it to reregister WebSphere MQ transport for SOAP with different .NET Framework versions.

**Note:** On Windows 2003 Server, you must also run the **aspnet\_regiis** utility, even if you are not deploying to Internet Information Server (IIS). The location of the **aspnet\_regiis.exe** utility might vary with different versions of the Microsoft .NET Framework, but it is typically located in: %SystemRoot%\Microsoft.NET\Framework\version number\aspnet\_regiis. If multiple versions are installed, use **aspnet\_regiis** for the version of .NET Framework you are using.

Parent topic: [Reference](#)

### Related tasks

[Installing WebSphere MQ Web transport for SOAP](#)

This build: January 26, 2011 11:09:36

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20120\_

## 1.9.6. Apache software license

Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>

<http://www.apache.org/licenses/>

Apache License  
Version 2.0, January 2004  
<http://www.apache.org/licenses/>

### TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

#### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical

transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
  - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
  - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
  - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
  - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and

may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[ ]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.


Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at


<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

**Parent topic:** [Reference](#)

 This build: January 26, 2011 11:10:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21201\_

## 1.9.7. MQMD SOAP settings

The WebSphere® MQ SOAP sender and WebSphere MQ SOAP listener create a message descriptor (**MQMD**). The topic describes the fields you must set in the MQMD if you create your own SOAP sender or listener.

## Purpose

The values set in the **MQMD** control the exchange of messages between the WebSphere MQ SOAP sender, the WebSphere MQ SOAP listener, and the SOAP client program. If you create your own SOAP sender or listener, follow the rules in [Table 1](#).

## Description

[Table 1](#) describes how the **MQMD** fields are set by the WebSphere MQ SOAP sender and WebSphere MQ SOAP listener. If you write your own sender or listener you must set these fields in accordance with the rules for exchanging messages. The WebSphere MQ SOAP listener conforms to typical WebSphere MQ message exchange protocols. If you write your own sender to work with the WebSphere MQ SOAP listeners, you can set different **MQMD** values.

In [Table 1](#), the values in the Setting column are organized as follows:

### Request, One way

Settings made by WebSphere MQ SOAP sender.

### Response, Report

Settings made by WebSphere MQ SOAP listener in response to WebSphere MQ SOAP sender request.

### ALL

Settings made by both the WebSphere MQ SOAP sender and WebSphere MQ SOAP listener.

### Customized sender

You can write your own sender. Typically, a customized sender overrides the standard report options.

Table 1. MQMD SOAP settings

Field name	Setting	Values
<i>StrucId</i>	<b>ALL</b> MQMD_STRUC_ID	'MD b b '1
<i>Version</i>	<b>ALL</b> MQMD_VERSION_2	2
<i>Report</i>	<b>ALL</b> MQRO_NONE + MQRO_NEW_MSG_ID + MQRO_COPY_MSG_ID_TO_CORREL_ID + MQRO_EXCEPTION + MQRO_EXPIRY + MQRO_DISCARD <b>Customized sender</b> See <a href="#">Customized report options</a>	52428800
<i>MsgType</i>	<b>Request</b> MQMT_REQUEST <b>Response</b> MQMT_REPLY <b>Report</b> MQMT_REPORT <b>One way</b> MQMT_DATAGRAM	<b>MQMT_REQUEST</b> 1 <b>MQMT_REPLY</b> 2 <b>MQMT_REPORT</b> 4 <b>MQMT_DATAGRAM</b> 8
<i>Expiry</i>	<b>Request, One way</b> Specified by Expiry option in URI. Defaults to MQEI_UNLIMITED. <b>Response</b> Value of Expiry in request message <b>Report</b> MQEI_UNLIMITED	<b>MQEI_UNLIMITED</b> -1
<i>Feedback</i>	<b>Request, Response, One way</b> MQFB_NONE. <b>Report</b> <ul style="list-style-type: none"> <li>Generated by queue manager - value set according to normal rules.</li> <li>Generated by WebSphere MQ SOAP Listener:           <ul style="list-style-type: none"> <li><b>MQRC_BACKOUT_THRESHOLD_REACHED</b> Backout threshold for multiple attempts exceeded.</li> <li><b>MQRCCF_MD_FORMAT_ERROR</b> The message is not recognized as having an MQRFH2 header.</li> <li><b>MQRC_RFH_PARM_MISSING</b> A required parameter, for example, SoapAction, in <b>MQRFH2</b> is missing.</li> <li><b>MQRC_RFH_FORMAT_ERROR</b> A basic integrity check of the <b>MQRFH2</b> failed, for example, internal lengths are corrupted.</li> <li><b>MQRC_RFH_ERROR</b> The <b>MQRFH2</b> passed an integrity check, but the body of the message is not set to MQFMT_NONE.</li> </ul> </li> </ul>	<b>MQFB_NONE</b> 0 <b>MQRC_BACKOUT_THRESHOLD_REACHED</b> 2362 <b>MQRCCF_MD_FORMAT_ERROR</b> 3023 <b>MQRC_RFH_PARM_MISSING</b> 2339 <b>MQRC_RFH_FORMAT_ERROR</b> 2421 <b>MQRC_RFH_ERROR</b> 2334
<i>Encoding</i>		Depends on environment

	<b>ALL</b> MQENC_NATIVE	
<i>CodedCharSetId</i>	<b>ALL</b> Set to UTF-8	1208
<i>Format</i>	<b>Request, Response, One way</b> MQFMT_RF_HEADER_2 <b>Report</b> <b>Queue manager reports</b> Follows WebSphere MQ rules <b>WebSphere MQ SOAP listener reports</b> Format of the original request message.	<b>MQFMT_RF_HEADER_2</b> "MQHRF2 "
<i>Priority</i>	<b>Request, One way</b> Specified by Priority option in URI. Defaults to MQPRI_PRIORITY_AS_Q_DEF. <b>Response, Report</b> Value of <i>Priority</i> in the request message.	<b>MQPRI_PRIORITY_AS_Q_DEF</b> -1
<i>Persistence</i>	<b>Request, One way</b> MQPER_PERSISTENCE_AS_Q_DEF. <b>Response, Report</b> Value of <i>Persistence</i> in the request message.	<b>MQPER_PERSISTENCE_AS_Q_DEF</b> 2
<i>MsgId</i>	<b>Request, One way</b> Generated by the queue manager. <b>Response, Report</b> The WebSphere MQ SOAP sender sets MQRO_NEW_MSG_ID and <i>MsgId</i> is generated	<b>Generated</b> Unique value generated by the queue manager
<i>CorrelId</i>	<b>Request, One way, Report</b> MQCI_NONE <b>Response, Report</b> The WebSphere MQ SOAP sender sets MQRO_COPY_MSG_ID_TO_CORREL_ID and the listener copies <i>MsgId</i> from the request message.	<b>MQCI_NONE</b> 0
<i>BackoutCount</i>	<b>ALL</b> Not used	0
<i>ReplyToQ</i>	<b>Request</b> Specified by replyDestination option in URI. Defaults to SYSTEM.SOAP.RESPONSE.QUEUE. <b>Response, One way, Report</b> Left blank	
<i>ReplyToQMgr</i>	<b>ALL</b> Field left blank	Generated by the queue manager; see <a href="#">Reply-to-queue and queue manager</a> .
<i>UserIdentifier</i>	<b>Request, One way, Report</b> Left blank <b>Response</b> Depends on the -i <i>passContext</i> option supplied to the listener, and the authority the listener is running under.	<b>Request, One way, Report</b> Generated by the queue manager; see <a href="#">UserIdentifier</a> <b>Response</b> <i>Variable</i>
<i>AccountingToken</i>	<b>ALL</b> MQACT_NONE	<b>MQACT_NONE</b> Null string or blanks Set by the queue manager; see <a href="#">AccountingToken</a>
<i>ApplIdentityData</i>	<b>ALL</b> None	Null string or blanks <sup>2</sup>
<i>PutApplType</i>	<b>ALL</b> MQAT_NO_CONTEXT	<b>MQAT_NO_CONTEXT</b> 0 Value generated by queue manager; see <a href="#">PutApplType (MQLONG)</a> .
<i>PutApplName</i>	<b>ALL</b> None	Value generated by queue manager; see <a href="#">PutApplName (MQLONG)</a> .
<i>PutDate</i>	<b>ALL</b> None	Value generated by queue manager; see <a href="#">PutApplDate (MOCHAR8)</a> .
<i>PutTime</i>	<b>ALL</b> None	Value generated by queue manager; see <a href="#">PutApplTime (MOCHAR8)</a> .
<i>ApplOriginData</i>	<b>ALL</b> None	Null string or blanks <sup>2</sup>
<i>GroupId</i>		Nulls

	<b>Request, One way, Report</b> MQGI_NONE <b>Response</b> Field is copied from the request message	
<i>MsgSeqNumber</i>	<b>Request, One way, Report</b> Not used <b>Response</b> Field is copied from the request message	Generated by the queue manager; see <a href="#">MQPUT options relating to messages in groups and segments of logical messages</a> .
<i>Offset</i>	<b>Request, One way, Report</b> Not used <b>Response</b> Field is copied from the request message	0
<i>MsgFlags</i>	<b>Request, One way, Report</b> MQMF_NONE <b>Response</b> Field is copied from the request message	MQMF_NONE 0 See <a href="#">MsgFlags (MQLONG)</a>
<i>OriginalLength</i>	<b>Request, One way, Response</b> MQOL_UNDEFINED <b>Report</b> Length of original request message	MQOL_UNDEFINED -1
<b>Notes:</b> <ol style="list-style-type: none"> <li>The symbol <i>b</i> represents a single blank character.</li> <li>The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.</li> </ol>		

### Customized report options

You can write your own SOAP sender and use it with the supplied listeners. Typically you might write a sender to change the choice of report options. The WebSphere MQ SOAP listeners support most combinations of report options, as described in the following lists.

- Report options supported by WebSphere MQ SOAP listeners:

- o MQRO\_EXCEPTION
- o MQRO\_EXCEPTION\_WITH\_DATA
- o MQRO\_EXCEPTION\_WITH\_FULL\_DATA
- o MQRO\_DEAD\_LETTER\_Q
- o MQRO\_DISCARD\_MSG
- o MQRO\_NONE
- o MQRO\_NEW\_MSG\_ID
- o MQRO\_PASS\_MSG\_ID
- o MQRO\_COPY\_MSG\_ID\_TO\_CORREL\_ID
- o MQRO\_PASS\_CORREL\_ID

- Report options supported by the queue manager:

- o MQRO\_COA
- o MQRO\_COA\_WITH\_DATA
- o MQRO\_COA\_WITH\_FULL\_DATA
- o MQRO\_COD
- o MQRO\_COD\_WITH\_DATA
- o MQRO\_COD\_WITH\_FULL\_DATA
- o MQRO\_EXPIRATION
- o MQRO\_EXPIRATION\_WITH\_DATA
- o MQRO\_EXPIRATION\_WITH\_FULL\_DATA

- The following report options are not supported by the WebSphere MQ SOAP listeners.

- o MQRO\_PAN
- o MQRO\_NAN

The behavior of WebSphere MQ SOAP listeners in response to combinations of MQRO\_EXCEPTION\_\* and MQRO\_DISCARD is described in [Table 2](#).

The notation MQRO\_EXCEPTION\_\* indicates the use of either MQRO\_EXCEPTION, MQRO\_EXCEPTION\_WITH\_DATA or MQRO\_EXCEPTION\_WITH\_FULL\_DATA.

Table 2. Listener behavior resulting from MQRO\_EXCEPTION\_\* and MQRO\_DISCARD settings

	MQRO_DISCARD enabled	MQRO_DISCARD not enabled
<b>MQRO_EXCEPTION_* enabled</b>	Default behavior. Report messages are automatically generated if necessary and the original request discarded. If a report message could not be returned to the response queue the report message is sent to the dead letter queue.	Report messages are automatically generated if necessary and the original message is sent to the dead letter queue. If the report message could not be returned to the response queue it is also be sent to the dead-letter queue. In this case there are therefore two dead letter queue entries for the failed request.
<b>MQRO_EXCEPTION_* not enabled</b>	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The original

message is not sent to the dead letter queue. No notification is returned that the client can inspect, and the original request message is lost.

request message is however written to the dead letter queue when a report would otherwise have been generated.

Parent topic: [Reference](#)

Related information  
[MQMD – Message descriptor](#)

This build: January 26, 2011 11:10:03

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts20990\_

### 1.9.8. MQRFH2 SOAP settings

The WebSphere MQ SOAP senders and listeners create an MQRFH2 with the following settings to send a SOAP message.

#### Purpose

The WebSphere MQ SOAP senders and listeners add a <usr> folder to the <mcd> and <jms> MQRFH2 folders created by WebSphere MQ JMS. The <usr> folder contains information required by the SOAP container in the target environment.

Syntax diagram



#### Notes:

1. Pad with blanks to a length that is a multiple of four.
2. targetService is required for .NET Framework 1 or 2, and not used on Axis 1.4.
3. soapAction is optional for .NET Framework 1 or 2, and not used on Axis 1.4.

#### Fixed parameters

Table 1. Field settings for a SOAP message in the fixed part of the MQRFH2 structure

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFH <b>b</b> ' <u>1</u>
Version	MQRFH_VERSION_2	2
StrucLength	Variable	Total length of header including all NameValueLength and NameValueData fields.
Encoding	MQENC_NATIVE	Depends on the environment
CodedCharSetId	(UTF-8)	1208
Format	MQFMT_NONE	' <b>b b b b b b b b</b> '
Flags	MQRFH_NONE	0
NameValueCCSID	(UTF-8)	1208

#### Note:

1. The symbol **b** represents a single blank character.

## Variable parameters

The variable portion of an WebSphere MQ SOAP **MQRFH2** header is an extension of the WebSphere MQ JMS **MQRFH2** header. It must contain an `<mcd>`, `<jms>` and a `<usr>` folder, and the folders must occur in that order. Each folder is contained within the `NameValueData` field of the variable part of the **MQRFH2** header.

### length

Length of the `mcd`, `jms` or `usr`, folder and any trailing blanks. *length* must be a multiple of four.

### mcd folder parameters

Queue manager defined following rules for WebSphere MQ JMS applications.

### jms folder parameters

Queue manager defined following rules for WebSphere MQ JMS applications.

### contentType

contentType always contains the string `text/xml; charset=utf-8`.

### endpointURL

See [URI syntax and parameters for Web service deployment](#).

### targetService

<sup>1</sup>On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=javaDemos.service.StockQuoteAxis`. If `targetService` is not specified, a service is loaded using the default Axis mechanism.

<sup>2</sup>On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the `targetService` parameter makes it possible for a single WebSphere MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

### soapAction

### transportVersion

transportVersion is always set to 1.

## Example

The example shows an **MQRFH2** and the following SOAP message. The lengths of the folders are shown in decimal.

**Note:** `&` in the URI is encoded as `&amp;`;

```
52464820 00000002 000002B0 00000001 RFHh 0002 1208 0001
000004B8 20202020 20202020 00000000 1208 h h h h h h h h 0000
000004B8 1208
32 <mcd>
  <Msd>jms_bytes</Msd>
</mcd> h
208 <jms>
  <Dst>queue://queue://SOAPJ.demos</Dst>
  <Rto>queue://WMQSOAP.DEMO.QM/SYSTEM.SOAP.RESPONSE.QUEUE</Rto>
  <Tms>1157388516465</Tms>
  <Cid>ID:0000000000000000000000000000000000000000000000000000000000000000</Cid>
  <Dlv>1</Dlv>
</jms>
400 <usr>
  <contentType>text/xml; charset=utf-8</contentType>
  <transportVersion>1</transportVersion>
  <endpointURL>
    jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
    &amp;connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)
    clientConnection=localhost%25289414%2529)
    clientChannel(TESTCHANNEL)
    &amp;replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
    &amp;initialContextFactory=com.ibm.mq.jms.NoJndi
  </endpointURL>
</usr>
<?xml version="1.0" encoding="UTF-8"?>•
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getQuote
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="soap.server.StockQuoteAxis_Wmq">
      <in0 xsi:type="xsd:string">XXX</in0>
    </ns1:getQuote>
  </soapenv:Body>
</soapenv:Envelope>
```


Parent topic: [Reference](#)

### Related information

[MQRFH2 – Rules and formatting header 2](#)

<sup>1</sup> Java service only

<sup>2</sup> .NET service only

 This build: January 26, 2011 11:10:06

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts21000\_





```

Client connection

|--clientConnection--(--localhost%25289414WMQSOAP.DEMO.QM%2529--)-->
>--clientChannel--(--TESTCHANNEL--)------|

Target service

|--&--targetService---StockQuoteDotNet.asmx-----|

```

### configFile parameters

#### testName

The name of the test. Use *testName* in the **runivt** command

#### testDescription

Documentation about the test

#### testCommand

The command executed by the **runivt** command to make the client request.

#### testResponse

The exact response string returned by the client request to the console. For the test to succeed *testResponse* must match the actual response.

#### testListener

The name of the WebSphere MQ SOAP listener that is started by **runivt** to process the SOAP request. `dotnet` and `JMSax` are synonyms for the supplied listeners, **amqwSOAPNETlistener** and **SimpleJavaListener**.

### Examples

Figure 1. run all the default tests

```
runivt
```

Figure 2. run a specific test from the default tests

```
runivt dotnet
```


Figure 3. run a set of custom tests

```
runivt -c mytests.txt
```

Parent topic: [Reference](#)

#### Related tasks

[Verifying the WebSphere MQ transport for SOAP](#)

 This build: January 26, 2011 11:09:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20380\_

## 1.9.10. Secure Web services over WebSphere MQ transport for SOAP

You might secure Web services that use the WebSphere MQ transport for SOAP in one of two ways. Either create an SSL channel between the client and server, or use Web services security.

### [SSL and the WebSphere MQ transport for SOAP](#)

The WebSphere® MQ transport for SOAP provides several SSL options that can be specified for use with client channel configured to run in SSL mode. The options differ between the .NET and Java environments. The WebSphere MQ SOAP senders and listeners process only the SSL options that are applicable to their particular environment. They ignore options that are not applicable.


### [SSL connection factory parameters in the WebSphere MQ Web services URI](#)

Add SSL options to the list of connection factory options in the WebSphere MQ Web services URI.

Parent topic: [Reference](#)

#### Related reference

[URI syntax and parameters for Web service deployment](#)

 This build: January 26, 2011 11:10:07

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21060\_

### 1.9.10.1. SSL and the WebSphere MQ transport for SOAP

The WebSphere® MQ transport for SOAP provides several SSL options that can be specified for use with client channel configured to run in SSL mode. The options differ between the .NET and Java environments. The WebSphere MQ SOAP senders and listeners process only the SSL options that are applicable to their particular environment. They ignore options that are not applicable.

The presence or absence of the `sslCipherSpec` option for .NET clients and the `sslCipherSuite` option for Java clients determines whether SSL is used or not. If the option is not specified in the URI then by default SSL is not used and all other SSL options are ignored. All SSL options are optional except where indicated.

For WebSphere MQ clients, set the SSL attributes in the URI or channel definition table. On the server, set the attributes using the facilities of WebSphere MQ.

By default, the standard WebSphere MQ SSL option, `SSLCAUTH`, is set when enabling SSL on the channel. Clients must authenticate themselves before SSL communication can commence. If `SSLCAUTH` is not set, SSL communications are established without client authentication.

To authenticate themselves, clients must have a certificate assigned in their key repository that is acceptable to the queue manager. For additional security, WebSphere MQ channels can be configured to accept only certificates from a restricted list. The list is restricted by checking the distinguished name of the certificate against the peer name attribute of the channel.

If you use Java, the first SSL connection from a WebSphere MQ SOAP client causes the following SSL parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined.

If you use .NET, the first SSL connection from a WebSphere MQ SOAP client causes the following SSL parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyRepository`
- `sslCryptoHardware`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined. These parameters are reset if all SSL connections become inactive and a new SSL connection is made.

The following properties can also be specified as system properties:


- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`

If they are specified both as system properties and in the URI, and the values differ, the deployment utility displays a warning. The URI values take precedence.

**Parent topic:** [Secure Web services over WebSphere MQ transport for SOAP](#)

#### Related reference

[SSL connection factory parameters in the WebSphere MQ Web services URI](#)

 This build: January 26, 2011 11:10:06

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21030\_

## 1.9.10.2. SSL connection factory parameters in the WebSphere MQ Web services URI

Add SSL options to the list of connection factory options in the WebSphere® MQ Web services URI.

### Purpose

You can use a secure connection between a WebSphere MQ Web services client and the queue manager hosting the web service. The SSL options control how SSL is configured on the WebSphere MQ client-server channel connection.

Syntax diagram

```
>>-----<<
SSL (Java)
|--sslCipherSuite--(--CipherSuite--)----->
>--sslKeyStore--(--KeyStoreName--)----->
>--sslKeyPassword--(--KeyStorePassword--)----->
>--sslTrustStore--(--TrustStore--)----->
>--sslTrustStorePassword--(--TrustStorePassword--)----->
>--| SSL (Common) |-----|
SSL (.NET)
|--sslCipherSpec--(--CipherSpec--)----->
>--sslKeyRepository--(--KeyRepository--)--| CryptoHardware |---->
>--| SSL (Common) |-----|
SSL (Common)
```





flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

#### **-d msec**

*msecs* specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

#### **-i Context**

*Context* specifies whether the listeners pass identity context. *Context* takes the following values:

##### **passContext**

Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the `MQOPEN` call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed `MQOPEN`. The listener then continues to process subsequent incoming messages as normal.

##### **ownContext**

The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

#### **-n numThreads**

*numThreads* specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

#### **-v**

-v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

#### **-w serviceDirectory**

*serviceDirectory* is the directory containing the web service.

#### **-x transactionality**

*transactionality* specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

##### **onePhase**

WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

##### **twoPhase**

Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

##### **none**

No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.


### Java Example

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.NoJndi"
-n 20
```

Parent topic: [Reference](#)

#### Related reference

[WebSphere MQ SOAP listeners](#)

 This build: January 26, 2011 11:09:37

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20150\_

## 1.9.12. WebSphere MQ SOAP listeners

A WebSphere® MQ SOAP listener reads an incoming SOAP request from the queue specified as the destination in the URI. It checks the format of the request message and then invokes a Web service using the Web services infrastructure. A WebSphere MQ SOAP listener returns any response or error from a Web service using the reply destination queue in the URI. It returns WebSphere MQ reports to the reply queue.

The term listener is used here in its standard Web services sense. It is distinct from the standard WebSphere MQ listener invoked by the **runmqisr** command.

### Description

The Java SOAP listener is implemented as a Java class and run services using Axis 1.4. The .NET listener is a console application and runs .NET Framework 1 or .NET Framework 2 services. For .NET Framework 3 services, use the WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF).

The deployment utility creates scripts to start Java or .NET SOAP listeners automatically. A SOAP Listener can be started manually using either the **amqSOAPNETListener** command, or by calling the `SimpleJavaListener` class. You can configure the WebSphere MQ SOAP listener to be started as a WebSphere MQ service by setting the -s option in the deployment utility. Alternatively, start listeners using triggering, or use the start and end listener scripts generated by the deployment utility. You can configure triggering manually, or use the -tmq and -tmp deployment options to configure triggering automatically. You can end a listener by setting the request queue to `GET(DISABLED)`.

Table 1. Command scripts generated by the deployment utility

Web service Infrastructure	UNIX	Windows Java	Windows .NET
Start listener	startWMQJListener.sh	startWMQJListener.cmd	startWMQNListener.cmd
Stop listener	endWMQJListener.sh	endWMQJListener.cmd	endWMQNListener.cmd



```

>-----|
|      .- -1---. | |      .-passContext-. |
|'- -d -+-msecs+-' | '- -i -+-ownContext---'
>-----|
|      .-10-----| | '- -v '
|'- -n -+-numThreads+-'
>-----|
|      .-onePhase-----| |
|'- -x -+-twophase-----| |
|      '-none-- -a -LowMsgIntegrity-'

```

## Required parameters

### URI platform

See [URI syntax and parameters for Web service deployment](#).

### -?

Print out help text describing how the command is used.

## Optional parameters

### -a *integrityOption*

*integrityOption* specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

#### DefaultMsgIntegrity

For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

#### LowMsgIntegrity

For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

#### HighMsgIntegrity

For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

### -d *msecs*

*msecs* specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

### -i *Context*

*Context* specifies whether the listeners pass identity context. *Context* takes the following values:

#### passContext

Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

#### ownContext

The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

### -n *numThreads*

*numThreads* specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

### -v

-v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

### -w *serviceDirectory*

*serviceDirectory* is the directory containing the web service.

### -x *transactionality*

*transactionality* specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

#### onePhase

WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

#### twoPhase

Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

#### none

No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

## .NET Example

```

amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.NoJndi"
-w C:/wmqsoap/demos

```




-n 20


**Java Example**

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi"
-n 20
```

Parent topic: [Reference](#)**Related reference**

[amqWSOAPNETListener: WebSphere MQ SOAP listener for .NET Framework 1 or 2](#)  
[SimpleJavaListener: WebSphere MQ SOAP Listener for Axis 1.4](#)

 This build: January 26, 2011 11:10:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)
 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

 This topic's URL:  
 ts20830\_
**1.9.13. WebSphere MQ transport for SOAP sender**

Sender classes are provided for Axis and .NET Framework 1 and .NET Framework 2. The sender constructs a SOAP request and puts it on a queue, it then blocks until it has read a response from the response queue. You can alter the behavior of the classes by passing different URIs from a SOAP client. For .NET Framework 3 use WebSphere® MQ custom channel for Microsoft Windows Communication Foundation (WCF).

**Purpose**

The WebSphere MQ SOAP sender puts a SOAP request to invoke a Web service onto a WebSphere MQ request queue. The sender sets fields in the **MQRFH2** header according to options specified in the URI, or according to defaults.

If you need to change the behavior of a sender beyond what is possible using the URI options, write your own sender. Your sender can work with the WebSphere MQ transport for SOAP listeners, or with other SOAP environments. Your sender must construct SOAP messages in the format defined by WebSphere MQ. The format is supported by the WebSphere MQ SOAP listener, and also SOAP listeners provided by WebSphere Application Server and CICS®. Your sender must follow the rules for an WebSphere MQ requestor. The WebSphere MQ SOAP listener returns reply and report messages. See [MQMD SOAP settings](#) for details how to set the report options in the **MQMD**. The report options control the report messages returned by the WebSphere MQ SOAP listener.

**Description**

The WebSphere MQ SOAP Java sender is registered with the Axis host environment for the `jms:` URI prefix. The sender is implemented in the class `com.ibm.mq.soap.transport.jms.WMQSender`, which is derived from `org.apache.axis.handlers.BasicHandler`. If the Axis host environment detects a `jms:` URI prefix it invokes the `com.ibm.mq.soap.transport.jms.WMQSender` class. The class blocks after placing the message until it has read a response from the response queue. If no response is received within a timeout interval the sender throws an exception. If a response is received within the timeout interval the response message is returned to the client using the Axis framework. Your client application must be able to handle these response messages.

For Microsoft® .NET Framework 1 and .NET Framework 2 services, the WebSphere MQ SOAP sender is implemented in the class `IBM.WMQSOAP.MQWebRequest`, which is derived from `System.Net.WebRequest` and `System.Net.IWebRequestCreate`. If .NET Framework 1 or .NET Framework 2 detects a `jms:` URI prefix it invokes the `IBM.WMQSOAP.MQWebRequest` class. The sender creates an `MQWebResponse` object to read the response message from the response queue and return it to the client.

`com.ibm.mq.soap.transport.jms.WMQSender` is a final class, and `IBM.WMQSOAP.MQWebRequest` is sealed. You cannot modify their behavior by creating subclasses.

**Parameters**

Set the URI to control the behavior of the WebSphere MQ SOAP sender in a Web service SOAP client. The deployment utility creates Web service client stubs incorporating the URI options supplied to the deployment utility.

**[Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender](#)**


A client connection channel definition is an alternative to setting connection properties in the `ConnectionFactory` attribute of the Web service URI. The connection properties are `clientChannel`, `clientConnection`, and `SSL` parameters.


Parent topic: [Reference](#)**Related concepts**

[Using WebSphere MQ custom channels for WCF](#)

**Related reference**

[MQMD SOAP settings](#)  
[URI syntax and parameters for Web service deployment](#)

 This build: January 26, 2011 11:09:38

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)
 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

 This topic's URL:  
 ts20340\_
**1.9.13.1. Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender**

A client connection channel definition is an alternative to setting connection properties in the `ConnectionFactory` attribute of the Web service URI. The connection properties are `clientChannel`, `clientConnection`, and `SSL` parameters.

## Description

Create the client channel description table by defining client connections. Even if a Web services client connects to different queue managers, create all the connections in the connection table on a single queue manager. The default name and location of the connection table is `queue_manager_directory/@ipcc/AMQCLCHL.TAB`.

Pass the location of the connection table to a Java client by setting the `com.ibm.mq.soap.transport.jms.mqchlurl` system property.

Pass the location of the connection table to a .NET client by setting the `MQCHLLIB` and `MQCHLTAB` environment variables.

You might provide both a channel connection table and channel connection parameters in the `ConnectionFactory` attribute of the Web service URI. The values set in the `ConnectionFactory` take precedence over the values in the channel definition table.

## Using a channel definition table in Java

Figure 1. Starting Java client using a configuration file

```
java -Dcom.ibm.msg.client.config.location=file:/C:/mydir/myjms.config MyAppClass
```

Figure 2. `myjms.config`

```
com.ibm.mq.soap.transport.jms.mqchlurl=file:/C:/ibm/wmq/qmgrs/QM1/@ipcc/AMQCLCHL.TAB
```

Parent topic: [WebSphere MQ transport for SOAP sender](#)

This build: January 26, 2011 11:10:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20860\_

## 1.9.14. Transactions

Use the `-x` option when starting the listener, to run Web services transactionally. Select the integrity of messages by setting the `persistence` option in the service URI.

### Web services

Use the `-x` option when starting the listener, to run Web services transactionally. On .NET Framework 1 and 2 the SOAP listener uses Microsoft Transaction Coordinator (MTS). On Axis 1.4, the SOAP listener uses queue manager coordinated transactions.

### Web service clients

The SOAP senders are not transactional.

### WebSphere MQ bindings

You can set the binding type for the SOAP sender. It can connect as a WebSphere MQ server application, or as a client application. You can also bind the SOAP sender as an XA-client on .NET.

### Message persistence

Select the level of persistence by setting the `Persistence` option in the URI.

### Web service transactions

You can use Web service transactions, because the SOAP sender is not transactional. If you write your own SOAP sender, and intend to use Web service transactions, do not create a transactional SOAP sender. You cannot send the request message and receive the reply message in the same transaction. The send and receive must not be coordinated by the Web service transaction.

Parent topic: [Reference](#)

This build: January 26, 2011 11:10:07

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21050\_

## 1.9.15. URI syntax and parameters for Web service deployment

The syntax and parameters to deploy a WebSphere MQ Web service are defined in a URI. The deployment utility generates a default URI based on the name of the Web service. You can override the defaults by defining your own URI as a parameter to the deployment utility. The deployment utility incorporates the URI in the generated Web service client stubs.

### Purpose

A web service is specified using a Universal Resource Identifier (URI). The syntax diagram specifies the URI that is supported in the WebSphere® MQ transport for SOAP. The URI controls over WebSphere MQ-specific SOAP parameters and options used to access target services. The URI is compatible with Web services hosted by .NET, Apache Axis 1, WebSphere Application Server and CICS®.

### Description

The URI is incorporated into the Web service client classes generated by the deployment utility. The client passes the URI to WebSphere MQ SOAP Sender in a WebSphere MQ message. The URI controls the processing performed by both the WebSphere MQ SOAP Sender and WebSphere MQ SOAP listener.

### Syntax

The URI syntax is as follows:

```
jms:/queue?name=value&name=value...
```

where **name** is a parameter name and **value** is an appropriate value, and the **name=value** element can be repeated any number of times with the second and subsequent occurrences being preceded by an ampersand (&).

Parameter names are case sensitive, as are names of WebSphere MQ objects. If any parameter is specified more than once, the final occurrence of the parameter takes effect. Client applications can override a generated parameter by appending another copy of the parameter to the URI. If any additional unrecognized parameters are included, they are ignored.

If you store a URI in an XML string, you must represent the ampersand character as `&amp;`. Similarly, if a URI is coded in a script, take care to escape characters such as `&` that would otherwise be interpreted by the shell.

Syntax diagram

```
>>-----<<
URI (.NET service)
                                     (1)
|--- -u -jms:--/--queue--?--destination---+--queue-----+----->
                                     .-SOAPN--.--className-----
>--&--targetService---serviceName--| URI (Common) |-----|
URI (Java service)
                                     (1)
|--- -u -jms:--/--queue---+--destination-----+----->
                                     .-Default Axis mechanism-----
>--&--targetService---serviceName--| URI (Common) |-----|
'&--targetService---serviceName-'
URI (Common)
|&--initialContextFactory---com.ibm.mq.jms.NoJndi----->
>--&--connectionFactory---+-(--)-+-----+----->
|                                     |
| V                                     |
|---| Name (value) |---+
>-----+----->
|                                     |
|&--persistence---+1+-----+----->
|                                     |
|&--priority---+priorityValue+-----+----->
|                                     |
|&--replyDestination---+replyQueue-----+----->
|                                     |
|&--timeout---+waitTime+-----+----->
|                                     |
|&--timeToLive---+expiryTime+-----+----->
Name (value)
|&--connectQueueManager---(+qMgrName+)-+----->
|                                     |
|                                     |
|                                     |
|&--binding---(+client-----+)-+----->
|                                     |
|&--server-----+-----+----->
|                                     |
|&--xaclient-----+----->
|                                     |
|&--clientChannel---(+clientChannelName+)-+----->
|                                     |
|&--clientConnection---(+IPV4Address+)-+----->
|                                     |
|                                     |
|&--IPV6Address+-----+----->
|&--Hostname-----+----->
+---| SSL (.NET) |---+
+---| SSL (Java) |---
```

#### Notes:

1. The queue manager transforms *className* to a queue name following the steps described in [Destination to queue name transformation](#)
2. **client** is the default if other options appropriate for a client are specified; for example **clientConnection**.
3. **xaclient** applies to .NET only

#### Destination to queue name transformation

1. *className* is prefixed with `SOAPJ.` for Java services or with `SOAPN.` for .NET services.

2. The file extension is removed from the full path name given in the *className* parameter.
3. The resulting string is truncated to no more than 48 characters
4. Directory separator characters are replaced with period characters.
5. Embedded spaces are replaced with underscore characters.
6. The colon following a drive prefix letter is replaced with a period for a .NET service.

**Note:** In some environments, a queue name generated by the deployment utility might not be unique. The deployment utility makes checks whether to create the queue. You might choose to override the deployment utility by restructuring the deployment directory hierarchy, or by customizing the supplied deployment process.

### Required URI parameters

#### **destination=queue**

*queue* is the name of the request destination. It can be a queue or a queue alias. If it is queue alias, the alias can resolve to a topic.

- If the -u parameter is omitted *queue* is generated from *classname* using the steps described in [Destination to queue name transformation](#).
- If the -u parameter is specified *queue* is required and must be the first parameter of the URI after the initial **xmlns:queue?** string. Specify either a WebSphere MQ queue name, or a queue name and queue manager name connected by an @ symbol, for example  
SOAPN.trandemos@WMQSOAP.DEMO.QM.
- The deployment utility checks whether the queue name, generated or provided, matches the name of an existing queue. The action taken is described in [Table 1](#).

Table 1. queue validation

Listener script exists?	Listener script exists in the ./generated/server directory		Listener script does not exist in the ./generated/server directory
Queue in listener script matches queue?	queue does not match request queue being used in listener script	queue matches request queue being used in listener script	
<b>queue exists</b>	<ul style="list-style-type: none"> <li>• Deployment exits with an error.</li> <li>• The service has already been deployed in ./generated/server, but using a different queue.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Deployment continues normally.</b></li> <li>• The service has already been deployed in ./generated/server</li> </ul>	<ul style="list-style-type: none"> <li>• Deployment exits with an error.</li> <li>• The listener startup script is not found in ./generated/server, but <i>queue</i> is in use by a different service or application.</li> </ul>
<b>queue does not exist</b>		<ul style="list-style-type: none"> <li>• <b>Deployment continues with a warning.</b></li> <li>• The previous deployment might have failed, because the startup is valid, but <i>queue</i> is missing.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>Deployment continues normally.</b></li> <li>• No service has been deployed from this directory.</li> </ul>

#### **&connectionFactory=Name(value)**

*Name* is one of the following parameters:

- [connectQueueManager](#)
- [binding](#)
- [clientChannel](#)
- [clientConnection](#)
- [SSL parameters](#)

See [Connection factory parameters](#) for a description of the values of these parameters.

#### **&targetService=serviceName**

<sup>1</sup>On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the `targetService` parameter makes it possible for a single WebSphere MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

### Optional URI parameters

#### **&initialContextFactory=contextFactory**

*contextFactory* is required and must be set to `com.ibm.mq.jms.NoJndi`. Make sure `NoJndi.jar` is in the class path for a WebSphere Application Server Web services client. `NoJndi.jar` returns Java objects based on the contents of the **connectionFactory** and **destination** parameters, rather than by reference to a directory.

#### **&targetService=serviceName**

<sup>2</sup>On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=javaDemos.service.StockQuoteAxis`. If `targetService` is not specified, a service is loaded using the default Axis mechanism.

#### **&persistence=messagePersistence**

*messagePersistence* takes one of the following values:

**0**

Persistence is inherited from the queue definition.

**1**

The message is non-persistent.

**2**

The message is persistent

#### **&priority=priorityValue**

*priorityValue* is in the range 0 - 9. 0 is low priority. The default value is environment-specific, which in the case of WebSphere MQ is 0.

#### **&replyDestination=*replyToQueue***

The queue at the client side to be used for the response message. The default reply queue is `SYSTEM.SOAP.RESPONSE.QUEUE`.

- Run the `setupWMQSOAP` script to create the default WebSphere MQ SOAP objects.
- Specify a model queue for *replyToQueue* to create either a temporary or permanent dynamic response queue. For both temporary and permanent dynamic response queues, a separate instance of dynamic queue is created for each request. If any of the following events happen the queue is deleted:
  - The response arrives and is processed.
  - The request times out.
  - The requesting program terminates.

For the best performance, use temporary dynamic queues rather than permanent dynamic queues. Do not send a persistent request message to a URI with a temporary dynamic queue. The WebSphere MQ listener SOAP fails to process the message and outputs an error. The client times out waiting for the reply.

- The `setupWMQSOAP` script creates a default permanent dynamic model queue called `SYSTEM.SOAP.MODEL.RESPONSE.QUEUE`.

#### **&timeout=*waitTime***

The time, in milliseconds, that the client waits for a response message. *waitTime* overrides values set by the infrastructure or client application. If not specified the application value, if specified, or infrastructure default is inherited.

**Note:** No relationship is enforced between timeout and timeToLive.

#### **&timeToLive=*expiryTime***

*expiryTime* is the time, specified in milliseconds, before the message expires. The default is zero, which indicates an unlimited lifetime.

**Note:** No relationship is enforced between timeout and timeToLive.

### Connection factory parameters

#### **connectQueueManager(*qMgrName*)**

*qMgrName* specifies the queue manager to which the client connects. The default is blank.

#### **binding(*bindingType*)**

*bindingType* specifies how the client is connected to *qMgrName*. The default is `auto`. *bindingType* takes the following values:

##### **auto**

The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are `clientConnection` or `clientChannel`.
2. Use a server connection.
3. Use a client connection.

Use **binding(auto)** in the *URI* if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

##### **client**

Use **binding(client)** in the *URI* to build a client configuration for the SOAP sender.

##### **server**

Use **binding(server)** in the *URI* to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the WebSphere MQ SOAP sender. Client type parameters are `clientConnection`, `clientChannel`, or `SSL` parameters.

##### **xaclient**

`xaclient` is applicable only on .NET and not for Java clients. Use an XA-client connection.

#### **clientChannel(*channel*)**

The SOAP client uses *channel* to make a WebSphere® MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. `clientChannel` is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the `MQCHLLIB` and `MQCHLTAB` environment variables; see [Use a channel definition table with the WebSphere\(r\) MQ SOAP transport for SOAP sender](#).

#### **clientConnection(*connection*)**

The SOAP client uses *connection* to make a WebSphere® MQ client connection. The default hostname is `localhost`, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

##### **IPV4 address**

For example, `192.0.2.0`.

##### **IPV6 address**

For example, `2001:DB8:0:0:0:0:0:0`.

##### **Host name**

For example, `www.example.com(1687)`.

#### **SSL platform**

See [SSL parameters](#)

### Sample URIs

#### **Note:**

1. `&` in the URI is encoded as `&amp;`
2. All the parameter listed previously are applicable to the clients.
3. Only **destination**, **connectionFactory** and **initialContextFactory** are applicable to the WCF service.

Figure 1. URI for an Axis service, supplying only required parameters

```
jms:/queue?destination=myQ&amp;connectionFactory=()&amp;initialContextFactory=com.ibm.mq.jms.NoJndi
```

Figure 2. URI for a .NET service, supplying only required parameters

```
jms:/queue?destination=myQ&amp;connectionFactory=()&amp;targetService=MyService.asmx
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 3. URI for an Axis service, supplying some optional connectionFactory parameters

```
jms:/queue?destination=myQ@myRQM&amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 4. URI for an Axis service, supplying the sslPeerName option of the connectionFactory parameter

```
jms:/queue?destination=myQ@myRQM&amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

### The Nojndi mechanism

The Nojndi mechanism enables JMS programs, which use JNDI interfaces, to use the same URI as WebSphere MQ programs, which do not use JNDI.


Parent topic: [Reference](#)

#### Related reference

[Secure Web services over WebSphere MQ transport for SOAP](#)  
[W3C SOAP over JMS URI for the WebSphere MQ Axis 2 client](#)

[1](#) .NET service only

[2](#) Java service only

 This build: January 26, 2011 11:09:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20640\_

## 1.9.15.1. The Nojndi mechanism

The Nojndi mechanism enables JMS programs, which use JNDI interfaces, to use the same URI as WebSphere® MQ programs, which do not use JNDI.

You can use the WebSphere MQ transport for SOAP to invoke Web services on WebSphere Application Server. WebSphere Application Server SOAP over JMS looks up the JMS resources using JNDI. The Web service client might be running on .NET, or using Axis 1.4, to invoke the Web service and not using JNDI. To use the same URL for the client and server, it must provide the same information whether the environment is using JNDI or not.

The URI passed to the WebSphere MQ transport for SOAP by a Web service client contains a specific WebSphere MQ queue manager and queue names. These names are parsed and used directly by WebSphere MQ SOAP support.

The Nojndi mechanism directs the initialContextFactory used by a JMS program to com.ibm.mq.jms.Nojndi. The com.ibm.mq.jms.Nojndi class is an implementation of the JNDI interface that returns the connectionFactory and destination from the URL as ConnectionFactory and Queue Java objects. If the JMS implementation is WebSphere MQ, MQConnectionFactory and MQQueue inherit from the ConnectionFactory and Queue classes.

By using the Nojndi mechanism, you are able to provide the same connection information to WebSphere Application server and .NET using the same URL.

Parent topic: [URI syntax and parameters for Web service deployment](#)

 This build: January 26, 2011 11:09:38

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20301\_

## 1.9.16. W3C SOAP over JMS URI for the WebSphere MQ Axis 2 client

Define a W3C SOAP over JMS URI to call a Web service from an Axis 2 client using WebSphere® MQ JMS as the SOAP transport. The Web service must be provided by a server that supports WebSphere MQ JMS and the W3C SOAP over JMS candidate recommendation for the SOAP/JMS binding.

### Description

The W3C candidate recommendation defines the SOAP over JMS binding; [SOAP over Java Message Service 1.0](#). Also useful for its examples is [URI Scheme for Java\(tm\) Message Service 1.0](#)<sup>1</sup>.

Use the syntax diagram to create W3C SOAP over JMS URIs that are syntactically correct, and are accepted by the WebSphere MQ Axis 2 client. It is limited to defining the URI that is accepted by the WebSphere MQ Axis 2 client. It is a subset of the W3C recommendation in two respects:

1. The jms-variant topic is not supported, and must not be specified in a URI passed to the WebSphere MQ Axis 2 client.
2. The following properties are omitted from the syntax diagram because they are JMS properties, and not part of the URI.
  - a. bindingVersion
  - b. contentType
  - c. soapAction
  - d. requestURI
  - e. isFault

The JMS properties are set by the Axis 2 client or the server.

The diagram extends the W3C recommendation by defining a custom parameter, `connectionFactory`. `connectionFactory` is used as an alternative to JNDI to specify how the Axis 2 client connects to a queue manager using a queue.

The WebSphere MQ Axis 2 client only accepts properties as part of the URI passed to the client by the client application or as environment variables. The WebSphere MQ Axis 2 client has no capability to process a WSDL document. The client application or a development tool might process the WSDL and create the URI to pass to the Axis 2 client. An WebSphere MQ Axis 2 client application cannot set the JMS message properties directly.

## Syntax

In accordance with the W3C recommendation, all the parameters can be obtained from environment variables. The environment variable names are formed by prefixing the parameter name with `soapjms_`. The syntax is: **soapjms\_parameterName**; for example,

```
set soapjms_targetServer=com.example.org.stockquote
```

If a parameter is set using an environment variable it overrides the value set in the URI.

In accordance with the W3C recommendation, all the parameters can be repeated. The last instance of a parameter is used, unless overridden by an environment variable.

```
jms-uri
>>-jms:--+queue--+--queueName--+-----+-----+>>
|
| | .-&-----+-----+
| | V |
| | '-?--+| Other properties |-----+--+
| | '-| Connection factory property |-
| | .-&-----+-----+
| | V | V |
| | '-jndi--+--destinationName--+?----| JNDI properties |-----+--+
| | '-| Other properties |-
|
Other properties
|-----+-----+
| | .-Server specific default-. |
| +targetService--+--serviceName-----+
| | .-PERSISTENT-----+
| +deliveryMode--+--NON_PERSISTENT--+
| | .-0-----+
| +timeToLive--+--milliseconds--+
| | .-4-----+
| +priority--+--1-----9--+
| | .- SYSTEM.SOAP.RESPONSE.QUEUE-. |
| +replyToName--+--replyToDestination-----+
| '-CustomParameter--+--customParameterValue-----'
|
JNDI properties
(1)
|-----+--jndiConnectionFactoryName--+--connectionFactoryName-----+
| +jndiInitialContextFactory--+--contextFactoryName-----+
| +jndiURL--+--providerURL-----+
| '+-----+
| '-jndi--+ContextParameter--+--contextParameterValue-'
|
Connection factory property
|-----+-----+
| | .-connectionFactory--+--( )-----+-----+
| | |
| | | V |
| | | .- -----+-----+
| | '+-connectionFactory--+--connectQueueManager--+--qMgrName--+--+--+
| | | +|- Binding |-----+
| | | '-| SSL property |-----+
|
Binding
|
| | .-----+-----+
| | V |
| | .- (--auto--)-----+-----+
| | |
| | | '-| Client binding property |-
| | | .-----+-----+
| | | V |
| | '+-binding--+--client--+-----+-----+
| | | '- (--server--)-----+-----+
|
Client binding property
|-----+-----+
| +clientChannel--+--channel--+-----+
|
>+-----+-----+
| | .-localhost-----+-----+
| | '-clientConnection--+--hostname-----+-----+
| | | +-IPV4 address+ | .-1414-----+
| | | '-IPV6 address-' | '- (--port number+--)-'
|
SSL property
|-----+-----+
| +sslCipherSuite--+--cipherSuite--+-----+
|
>+-----+-----+
| '-sslPeerName--+--peerName--+-----+
|
>+-----+-----+
| +sslKeyStore--+--KeyStoreName--+--sslKeyStorePassword--+--KeyStorePassword--+
|
>+-----+-----+
| '-sslTrustStore--+--TrustStoreName--+--sslTrustPassword--+--TrustStorePassword--+
```

```

>-----+----->
'-sslKeyResetCount--(--byteCount--)-'
>-----+----->
'-sslFipsRequired--(--fipsCertified--)-'
>-----+-----|
'-sslLDAPCRLServers--(--LDAPServerList--)-'

```

**Notes:**

1. **jndiConnectionFactoryName**, **jndiConnectionFactoryName** and **jndiURL** are all required parameters. **jndi**—*ContextParameter* is optional.

**Parameters****connectionFactory=connectionFactoryParameterList**

*connectionFactoryParameterList* are parameters that qualify how the Axis 2 client connects to a queue manager when the destination variant is queue.

connectionFactory must not be specified with the jndi destination variant.

The parameters are not passed to the server in the request URI.

If connectionFactory is omitted, the queue must belong to a default queue manager running on the same server as the Axis 2 client.

The *connectionFactoryParameterList*:

**binding(bindingType)**

*bindingType* specifies how the client is connected to *qMgrName*. The default is auto. *bindingType* takes the following values:

**auto**

The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are clientConnection or clientChannel.
2. Use a server connection.
3. Use a client connection.

Use **binding(auto)** in the *URI* if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

**client**

Use **binding(client)** in the *URI* to build a client configuration for the SOAP sender.

**server**

Use **binding(server)** in the *URI* to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the WebSphere MQ SOAP sender. Client type parameters are clientConnection, clientChannel, or SSL parameters.

**xaclient**

xaclient is applicable only on .NET and not for Java clients. Use an XA-client connection.

**clientChannel(channel)**

The SOAP client uses *channel* to make a WebSphere® MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. clientChannel is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the MQCHLLIB and MQCHLTAB environment variables; see [Use a channel definition table with the WebSphere\(r\) MQ SOAP transport for SOAP sender](#).

**clientConnection(connection)**

The SOAP client uses *connection* to make a WebSphere® MQ client connection. The default hostname is localhost, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

**IPV4 address**

For example, 192.0.2.0.

**IPV6 address**

For example, 2001:DB8:0:0:0:0:0:0.

**Host name**

For example, www.example.com(1687).

**sslCipherSuite(CipherSuite)**

*CipherSuite* specifies the sslCipherSuite used on the channel. The CipherSuite specified by the client must match the CipherSuite specified on the server connection channel.

**sslFipsRequired(fipsCertified)**

*fipsCertified* specifies whether *CipherSpec* or *CipherSuite* must use FIPS-certified cryptography in WebSphere MQ on the channel. The effect of setting *fipsCertified* is the same as setting the *FipsRequired* field of the **MQSCO** structure on an **MQCONN** call.

**sslKeyStore(KeyStoreName)**

*KeyStoreName* specifies the sslKeyStoreName used on the channel. The keystore holds the private key of the client used to authenticate the client to the server. The key store is optional if the SSL connection is configured to accept anonymous client connections.

**sslKeyResetCount(bytecount)**

*bytecount* specifies the number of bytes passed across an SSL channel before the SSL secret key must be renegotiated. To disable the renegotiation of SSL keys omit the field or set it to zero. Zero is the only value supported in some environments, see [Renegotiating the secret key in WebSphere\(r\) MQ classes for Java\(tm\)](#). The effect of setting sslKeyResetCount is the same as setting the *KeyResetCount* field in the **MQSCO** structure on an **MQCONN** call.

**sslKeyStorePassword(KeyStorePassword)**

*KeyStorePassword* specifies the sslKeyStorePassword used on the channel.

**sslLDAPCRLServers(LDAPServerList)**

*LDAPServerList* specifies a list of LDAP servers to be used for Certificate Revocation List checking. the sslCryptoHardware used on the channel.



For SSL enabled client connections, *LDAPServerList* is a list of LDAP servers to be used for Certificate Revocation List (CRL) checking. The certificate provided by the queue manager is checked against one of the listed LDAP CRL servers; if found, the connection fails. Each LDAP server is tried in turn until connectivity is established to one of them. If it is impossible to connect to any of the servers, the certificate is rejected. Once a connection has been successfully established to one of them, the certificate is accepted or rejected depending on the CRLs present on that LDAP server.

If *LDAPServerList* is blank, the certificate belonging to the queue manager is not checked against a Certificate Revocation List. An error message is displayed if the supplied list of LDAP URIs is not valid. The effect of setting this field is the same as that of including *MQAIR* records and accessing them from an *MQSCO* structure on an *MQCONN*.

#### **sslPeerName(peerName)**

*peerName* specifies the *sslPeerName* used on the channel. See [Channel parameters - SSLPeer](#).

#### **sslTrustStore(TrustStoreName)**

*TrustStoreName* specifies the *sslTrustStoreName* used on the channel. The trust store holds the public certificate of the server, or its key chain, to authenticate the server to the client. The truststore is optional if the root certificate of a certificate authority is used to authenticate the server. In Java, root certificates are held in the JRE certificate store, *cacerts*.

#### **sslTrustStorePassword(TrustStorePassword)**

*TrustStorePassword* specifies the *sslTrustStorePassword* used on the channel.

#### **CustomParameter=customParameterValue**

*CustomParameter* is the user-defined name of a custom parameter, and *customParameterValue* is the value of the parameter.

Custom parameters that are not used by the Axis 2 client are sent by the Axis 2 client to the SOAP server. Consult the server documentation. *connectionFactory* is a custom parameter that is used by the Axis 2 client and is not passed to the server.

*CustomParameter* must not match the name of an existing parameter.

If *CustomParameter* starts with the string *jndi-* it is used in looking up a JNDI destination; see [jndi-](#).

#### **deliveryMode=deliveryMode**

*deliveryMode* sets the message persistence. The default is *PERSISTENT*.

#### **jndi:destinationName**

*destinationName* is a JNDI destination name that maps to a JMS queue. If the *jndi* destination variant is specified, you must provide a *destinationName*.

#### **jndiConnectionFactoryName=connectionFactoryName**

*connectionFactoryName* sets the JNDI name of the connection factory. If the destination variant is *jndi*, *connectionFactoryName* must be provided.

#### **jndiInitialContextFactory=contextFactoryName**

*contextFactoryName* sets the JNDI name of the initial context factory. If the destination variant is *jndi*, *contextFactoryName* must be provided. See [Using JNDI to retrieve administered objects in a JMS application](#).

#### **jndiURL=providerURL**

*jndiURL* sets the URL name of the JNDI provider. If the destination variant is *jndi*, *jndiURL* must be specified.

#### **jndi-ContextParameter=contextParameterValue**

*jndi-ContextParameter* is the user-defined name of a custom parameter that used to pass information to the JNDI provider. *contextParameterValue* is the information that is passed.

#### **priority=priorityValue**

*priorityValue* sets the JMS message priority. 0 is low, 9 is high. The default value is 4.

#### **queue:queueName**

*queueName* is the name of a JMS queue on which the SOAP request is placed. If the queue variant is specified, a queue name must be provided. If the queue does not belong to a default queue manager on the same server as the client, set the [connectionFactory](#) parameter.

#### **replyToName=replyToDestination**

*replyToDestination* sets the destination queue name. If the destination variant is *jndi*, the name is a JNDI name that must map to a queue. If the variant is *queue* the name is a JMS queue. The default value is *SYSTEM.SOAP.RESPONSE.QUEUE*.

#### **targetService=serviceName**

The name used by the SOAP server to start the target Web service.

On Axis, *serviceName* is the fully qualified name of a Java service, for example: *targetService=www.example.org.StockQuote*. If *targetService* is not specified, a service is loaded using the default Axis mechanism.

#### **timeToLive=milliseconds**

Set *milliseconds* to the time before the message expires. The default, 0, is the message never expires.

### Examples

Figure 1. Use *jms:jndi* to send a SOAP/JMS request

```
jms:jndi:REQUESTQ
?jndiURL=file:/C:/JMSAdmin
&jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
&jndiConnectionFactoryName=ConnectionFactory
&replyToName=RESPONSEQ
&deliveryMode(NON_PERSISTENT)
```

Figure 2. Use *jms:queue* to send a SOAP/JMS request

```
jms:queue:SOAPJ.demos
?connectionFactory=connectQueueManager(QM1)
Bind(Client)
ClientChannel(SOAPClient)
ClientConnection(www.example.org(1418))
&deliveryMode(NON_PERSISTENT)
```

Parent topic: [Reference](#)


#### Related tasks

[Connect an Axis2 client to a JAX-WS service using W3C SOAP over JMS and WebSphere Application Server](#)

#### Related reference

## [URI syntax and parameters for Web service deployment](#)

<sup>1</sup> Look for *URI Scheme for JMS*, in the W3C specification references, for the latest draft.

 This build: January 26, 2011 11:09:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts20642\_

### 1.9.17. Supported Web services

Code that has been written to run as a Web service does not need to be modified to use the WebSphere® MQ transport for SOAP. You do need to deploy services differently to run with the WebSphere MQ transport for SOAP rather than using HTTP.

#### Description

The WebSphere MQ transport for SOAP provides a SOAP listener to run services for .NET Framework 1 and .NET 2, and for Axis 1.4. The WebSphere MQ custom channel for Microsoft Windows Communication Foundation runs services for .NET Framework 3. WebSphere Application Server and CICS® provide support for running services over WebSphere MQ transport for SOAP. Create a custom Export to use WebSphere Enterprise Service Bus or WebSphere Process Server.

The WebSphere MQ SOAP listener can process SOAP requests transactionally. Run **amqwdeployWMQService** using the `-x` option. The two-phase option is only supported for listeners using server bindings. Other environments might provide transactional support for the WebSphere MQ transport for SOAP. Consult their documentation.

WebSphere MQ transport for SOAP currently does not support the emerging industry standard SOAP over JMS protocol that has been submitted to W3C. You can distinguish a SOAP/JMS message written to the new standard by looking for the JMS BindingVersion property. WebSphere MQ transport for SOAP does not set the BindingVersion property.

#### Axis 1.4

A Java class can typically be used without modification. The types of any arguments to the methods in the web service must be supported by the Axis engine. Refer to Axis documentation for further details. If the service uses a complex object as an argument, or returns one, that object must comply to the Java bean specification. See the examples in [Figure 3](#), [Figure 4](#), and [Figure 5](#):

1. Have a public parameter-less constructor.
2. Any complex types of the bean must have public getters and setters of the form:

```
>>+-get-+---complex-type-----<<
      '-set-'
```

Prepare the service for deployment the service using the **amqwdeployWMQService** utility. The service is invoked by the WebSphere MQ SOAP listener which uses `axis.jar` to run the service.

The only two-phase transaction manager supported for Axis 1.4 is WebSphere MQ.

The supplied deployment utility does not support the case where a service returns an object in a different package to the service itself. To use an object returned in a different package, write your own deployment utility. You can base your deployment utility on the supplied sample, or capture the commands it produces, using the `-v` option. Amend the commands to produce a tailored script.

If the service uses classes that are external to the Axis infrastructure and the WebSphere MQ SOAP run time environment, you must set the correct CLASSPATH. To change CLASSPATH, amend the generated script that starts or defines the listeners to include the services required, in one of the following ways:

- Amend the CLASSPATH directly in the script after the call to **amqwsetcp**.
- Create a service-specific script to customize the CLASSPATH and invoke this script in the generated script after the call to **amqwsetcp**.
- Create a customized deployment process to customize the CLASSPATH in the generated script automatically.

#### .NET Framework 1 and .NET Framework 2

A service that has already been prepared as an HTTP Web service does not need to be modified for use as a WebSphere MQ Web service. It needs to be deployed using the **amqwdeployWMQService** utility.

The only two-phase transaction manager supported for .NET Framework 1 and .NET 2 is Microsoft Transaction Server (MTS).

If the service code has not been prepared as an HTTP Web service you must convert it to a Web service. Declare the class as a web service and identify how the parameters of each method are formatted. You must check that any arguments to the methods of the service are compatible with the environment. [Figure 1](#) and [Figure 2](#) show a .NET class that has been prepared as a web service. The additions made are shown in bold type.

[Figure 1](#) uses the code-behind programming model for a .NET Web service. In the code-behind model, the source for the service is separated from the `.asmx` file. The `.asmx` file declares the name of the associated source file with the Codebehind keyword. WebSphere MQ has samples of both inline and code-behind .NET Web services.

.NET Web services source must be compiled before deployment by the **amqwdeployWMQService** deployment utility. The service is compiled into a library (`.dll`). The library must be placed in the `./bin` subdirectory of the deployment directory.

#### .NET Framework 3

Create a WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF) to invoke services deployed to .NET Framework 3. See [WebSphere MQ custom channel for Microsoft Windows Communication Foundation \(WCF\)](#) for a description of how to configure WCF to use the WebSphere MQ transport for SOAP.

#### WebSphere Application Server

You can invoke Web services hosted by WebSphere Application Server using the WebSphere MQ Transport for SOAP; see [Using SOAP over JMS to transport Web services](#).

You need to modify the WSDL generated by deployment of a JMS service to WebSphere Application Server in order to generate a Web services client. The WSDL created by deployment to WebSphere Application Server includes a URI with a JNDI reference to the JMS `InitialContextFactory`. You need to modify the JNDI reference to `Nojndi` and provide connection attributes as described in [URI syntax and parameters for Web service deployment](#).

### CICS

You can invoke CICS applications using the WebSphere MQ Transport for SOAP; see [Configuring your CICS system for Web services](#).

### WebSphere Enterprise Service Bus and WebSphere Process Server for Multiplatforms

WebSphere ESB and WebSphere Process Server for Multiplatforms support SOAP over JMS, with a ready built binding, only when using the default WebSphere Application Server messaging provider. Create a custom binding for JMS to support WebSphere MQ transport for SOAP. See [JMS data bindings](#) and [Web services with SOAP over JMS in IBM® WebSphere Process Server or IBM WebSphere Enterprise Service Bus, Part 2: Using the IBM WebSphere MQ JMS provider](#).

### Example

Figure 1. Service definition for .NET Framework 2: `Quote.asmx`

```
<%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
```

Figure 2. Service implementation for .NET Framework 2: `Quote.asmx.cs`

```
<%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace Quote {
    [WebService(Namespace = "http://www.example.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class QuoteDotNet : System.Web.Services.WebService {
        [WebMethod]
        public string getQuote(String symbol){
            return symbol.ToUpper();
        }
    }
}
```

Figure 3. Java JAX-RPC service interface using a complex type

```
package org.example.www;
public interface CustomerInfoInterface extends java.rmi.Remote {
    public org.example.www.CustomerRecord
        getCustomerName(org.example.www.CustomerRecord request)
        throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg;
}
```

Figure 4. Java JAX-RPC service implementation using a complex type

```
package org.example.www;
public class CustomerInfoPortImpl implements org.example.www.CustomerInfoInterface{
    public org.example.www.CustomerRecord
        getCustomerName(org.example.www.CustomerRecord request)
        throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg {
        request.setName(request.getID().toString());
        return request;
    }
}
```

Figure 5. Java JAX-RPC service bean implementation of a complex type

```
package org.example.www;
public class CustomerRecord {
    private java.lang.String name;
    private java.lang.Integer ID;
    public CustomerRecord() {}
    public java.lang.String getName() {
        return name;
    }
    public void setName(java.lang.String name) {
        this.name = name;
    }
    public java.lang.Integer getID() {
        return ID;
    }
    public void setID(java.lang.Integer ID) {
        this.ID = ID;
    }
}
```

Parent topic: [Reference](#)

### Related tasks

[Deploying a service to Axis 1.4 to use for WebSphere transport for SOAP using `amqwdployWMOService`](#)

[Deploying a service to .NET Framework 1 or 2 service to use WebSphere MQ transport for SOAP](#)

[Deploying a service to CICS Transaction Server to use WebSphere Transport for SOAP](#)

[Deploying a service to WebSphere Application Server to use WebSphere Transport for SOAP](#)


[Deploying a service to WebSphere ESB and Process Server service endpoint to use WebSphere Transport for SOAP](#)

[Developing Web services for WebSphere MQ transport for SOAP](#)

### Related reference

[Sample JAX-RPC Java service](#)

[Sample .NET services](#)

 This build: January 26, 2011 11:09:52

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts20610\_

## 1.9.18. WebSphere MQ transport for SOAP Web service clients

You can reuse an existing SOAP over HTTP client with WebSphere MQ transport for SOAP. You must make some small modifications to the code and build process to convert the client to work with WebSphere MQ transport for SOAP.

### Coding

JAX-RPC clients must be written in Java. .NET Framework 1 and 2 clients can be written in any language that uses the Common Language Runtime. Code examples are provided in C# and Visual Basic.

The level of transactional support depends on the client environment and the pattern of the SOAP interaction. The SOAP request and SOAP reply can not be part of the same atomic transaction.

You must call `IBM.WMQSOAP.Register.Extension()` in a .NET Framework 1, .NET Framework 2 client. In a JAX-RPC Java Web service client call `com.ibm.mq.soap.Register.extension` to register the WebSphere MQ SOAP sender. The method registers the WebSphere MQ transport for SOAP sender as the handler for SOAP messages using the `.jms:` protocol.

To create a .NET Framework 3 client, generate a Windows Communication Foundation client proxy using the `svcutil` tool; see [Generating a WCF client proxy and application configuration files using the svcutil tool with metadata from a running service](#).

### Libraries required to build and run .NET Framework 1 and 2 clients

- amqsoap
- System
- System.Web.Services
- System.Xml

### Libraries required to build and run Axis 1.4 clients

- `MQ_Install\java\lib\com.ibm.mq.soap.jar;`
- `MQ_Install\java\lib\com.ibm.mq.commonservices.jar;`
- `MQ_Install\java\lib\soap\axis.jar;`
- `MQ_Install\java\lib\soap\jaxrpc.jar`
- `MQ_Install\java\lib\soap\saa.jar;`
- `MQ_Install\java\lib\soap\commons-logging-1.0.4.jar;`
- `MQ_Install\java\lib\soap\commons-discovery-0.2.jar;`
- `MQ_Install\java\lib\soap\wsdl4j-1.5.1.jar;`
- `MQ_Install\java\jre\lib\xml.jar;`
- `MQ_Install\java\lib\soap\servlet.jar;`
- `MQ_Install\java\lib\com.ibm.mq.jar;`
- `MQ_Install\java\lib\com.ibm.mq.headers.jar;`
- `MQ_Install\java\lib\com.ibm.mq.pcf.jar;`
- `MQ_Install\java\lib\com.ibm.mq.jmqi.jar;`
- `MQ_Install\java\lib\com.ibm.mq.jmqi.remote.jar;`
- `MQ_Install\java\lib\com.ibm.mq.jmqi.local.jar;`
- `MQ_Install\java\lib\connector.jar;`
- `MQ_Install\java\lib\jta.jar;`
- `MQ_Install\java\lib\jndi.jar;`
- `MQ_Install\java\lib\ldap.jar`

### Register SOAP extension

```
>>+--+| Java |-----+-----<<
      +-| C# |-----+
      '-| Visual Basic |-'

Java

|--com.ibm.mq.soap.Register.extension()-----|

C#

|--IBM.WMQSOAP.Register.Extension();-----|

Visual Basic

|--IBM.WMQSOAP.Register.Extension-----|
```

### Client examples

[Figure 1](#) is an example of a .NET Framework 1 or .NET Framework 2 C# client that uses the inline programming model. The `IBM.WMQSOAP.Register.Extension()` method registers the WebSphere MQ SOAP sender with .NET as the `.jms:` protocol handler.

Figure 1. C# Web service client sample

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
            }
        }
    }
}
```

```
        Console.WriteLine("Response is: " + q.getQuote("ibm"));
    } catch (Exception e) {
        Console.WriteLine("Exception is: " + e);
    }
}
}
```

[Figure 2](#) is an example of an Java client that uses the JAX-RPC static proxy client interface. The `com.ibm.mq.soap.Register.extension()` method registers the WebSphere MQ SOAP sender with the service proxy to handle the `jms: protocol`.

Figure 2. Java Web service client example

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
    public static void main(String[] args) {
        try {
            Register.extension();
            QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
            System.out.println("Response = "
                + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
        } catch (Exception e) {
            System.out.println("Exception = " + e.getMessage());
        }
    }
}
```

**Parent topic:** [Reference](#)

#### Related tasks

[Deploying a Web service client to Axis 1.4 to use WebSphere MQ transport for SOAP](#)


[Deploying a Web service client to .NET Framework 1 and 2 to use WebSphere MQ transport for SOAP](#)

[Developing Web services for WebSphere MQ transport for SOAP](#)


#### Related reference

[Sample JAX-RPC Java clients](#)

[Sample .NET clients](#)

 This build: January 26, 2011 11:09:51

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts20550\_

## 2. WebSphere MQ bridge for HTTP

With WebSphere® MQ bridge for HTTP, client applications can exchange messages with WebSphere MQ without the need to install a WebSphere MQ client. You can call WebSphere MQ from any platform or language with HTTP capabilities.

### Introduction to WebSphere MQ bridge for HTTP

The WebSphere MQ bridge for HTTP is a Java, Enterprise Environment (JEE) Web application. HTTP clients can send **POST**, **GET**, and **DELETE** requests to it to put, browse and delete messages from WebSphere MQ queues. The WebSphere MQ bridge for HTTP is not suitable for use with messages, if guaranteed delivery is required.

### Installing, configuring, and verifying WebSphere MQ bridge for HTTP

Obtain WebSphere MQ bridge for HTTP by installing "Java Messaging and Web Services" from either the WebSphere MQ client or server installation materials. Deploy WebSphere MQ bridge for HTTP to a suitable application server.

### Publish/subscribe using the WebSphere MQ bridge for HTTP

WebSphere MQ bridge for HTTP uses the WebSphere MQ classes for JMS publish/subscribe interface. HTTP **POST** creates a publication. HTTP **DELETE** creates a non-durable managed subscription. You must configure publish/subscribe for JMS before using the topic URI.

### Running the WebSphere MQ bridge for HTTP samples

The WebSphere MQ bridge for HTTP samples are available for use on only the Windows operating system. The samples show you how to submit HTTP **POST** and HTTP **DELETE** commands to WebSphere MQ bridge for HTTP from Java programs.


### Security considerations for WebSphere bridge for HTTP

Standard Web security considerations apply to authenticating a Web browser client. Authorization to WebSphere MQ resources is at the level of the user running the WebSphere Bridge for HTTP servlet, and not the individual Web browser client. Standard WebSphere MQ security consideration apply to WebSphere MQ.


### Reference material for WebSphere MQ bridge for HTTP

Reference topics for WebSphere MQ bridge for HTTP, arranged alphabetically

**Parent topic:** [WebSphere MQ transport for SOAP and WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts12000\_

## 2.1. Introduction to WebSphere MQ bridge for HTTP

The WebSphere® MQ bridge for HTTP is a Java, Enterprise Environment (JEE) Web application. HTTP clients can send **POST**, **GET**, and **DELETE** requests to it to put, browse and delete messages from WebSphere MQ queues. The WebSphere MQ bridge for HTTP is not suitable for use with messages, if guaranteed delivery is required.

### Benefits

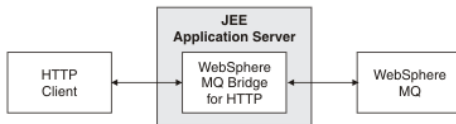
With the WebSphere MQ bridge for HTTP you can send and receive WebSphere MQ messages using HTTP from a wide variety of environments:

- Environments that support HTTP, but not WebSphere MQ.
- Environments that have insufficient storage space to install a WebSphere MQ client.
- Environments that are too numerous to install the WebSphere MQ client on each system that requires access to WebSphere MQ.
- Web-based applications from which you want to send or receive messages without coding your own bridge to WebSphere MQ.
- Web-based applications that you want to enhance, using asynchronous techniques such as AJAX. WebSphere MQ bridge for HTTP makes WebSphere MQ queues and topics available using Representation State Transfer (REST) over HTTP.

HTTP support can be used with both point-to-point and publish/subscribe messaging topologies.

### How does HTTP support work?

Figure 1. WebSphere MQ bridge for HTTP



The WebSphere MQ bridge for HTTP Web application receives HTTP requests from one or more clients. It interacts with WebSphere MQ on their behalf, and returns HTTP responses to them.

The WebSphere MQ bridge for HTTP is a JEE servlet that is connected to WebSphere MQ using a resource adapter. The HTTP servlet handles three different types of HTTP requests: **POST**, **GET**, and **DELETE**.

Table 1. WebSphere MQ bridge for HTTP verbs

HTTP Request	Result
POST	Puts a message on a queue or topic.
GET	Browses the first message on a queue. In line with the HTTP protocol, <b>GET</b> does not delete the message from the queue. Do not use <b>GET</b> with publish/subscribe messaging.
DELETE	Gets and deletes a message from a queue or topic.

### HTTP POST example

HTTP **POST** puts a message to a queue, or a publication to a topic. The **HTTPPOST** Java sample is an example an HTTP **POST** request of a message to a queue. Instead of using Java, you could create an HTTP **POST** request using a browser form, or an AJAX toolkit instead.

[Figure 2](#) shows an HTTP request to put a message on a queue called `myQueue`. This request contains the HTTP header `x-msg-correlId` to set the correlation ID of the WebSphere MQ message.

Figure 2. Example of an HTTP **POST** request to a queue

```

POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlId: 1234567890
Content-Length: 50

Here's my message body that will appear on the queue.
  
```

[Figure 3](#) shows the response sent back to the client. There is no response content.

Figure 3. Example of an HTTP **POST** response

```

HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
  
```

### HTTP DELETE example

HTTP **DELETE** gets a message from a queue and deletes the message, or retrieves and deletes a publication. The **HTTPDELETE** Java sample is an example an HTTP **DELETE** request reading a message from a queue. Instead of using Java, you could create an HTTP **DELETE** request using a browser form, or an AJAX toolkit instead.

[Figure 4](#) is an HTTP request to delete the next message on queue called `myQueue`. In response, the message body is returned to the client. In WebSphere MQ terms, HTTP **DELETE** is a destructive get.

The request contains the HTTP request header `x-msg-wait`, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

Figure 4. Example of an HTTP **DELETE** request

```

DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlId
  
```

[Figure 5](#), is the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

Figure 5. Example of an HTTP **DELETE** response

```

HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890
  
```

```
Here's my message body that will appear on the queue.
```

### HTTP GET example

HTTP **GET** gets a message from a queue. The message remains on the queue. In WebSphere MQ terms, HTTP **GET** is a browse request. You could create an HTTP **DELETE** request using a Java client, a browser form, or an AJAX toolkit.

[Figure 6](#) is an HTTP request to browse the next message on queue called `myQueue`.

The request contains the HTTP request header `x-msg-wait`, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

*Figure 6. Example of an HTTP GET request*

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

[Figure 7](#) is the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

*Figure 7. Example of an HTTP GET response*

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890
```

```
Here's my message body that will appear on the queue.
```

Parent topic: [WebSphere MQ bridge for HTTP](#)

#### Related tasks

[Running the WebSphere MQ bridge for HTTP samples](#)

#### Related reference

[Reference material for WebSphere MQ bridge for HTTP](#)

This build: January 26, 2011 11:10:15

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

s11370\_

## 2.2. Installing, configuring, and verifying WebSphere MQ bridge for HTTP

Obtain WebSphere® MQ bridge for HTTP by installing "Java Messaging and Web Services" from either the WebSphere MQ client or server installation materials. Deploy WebSphere MQ bridge for HTTP to a suitable application server.

### Before you begin

Check the prerequisite products at [WebSphere MQ requirements](#). The installation process does not check for the presence and availability of the prerequisite software for running WebSphere MQ bridge for HTTP. You must verify that the prerequisites are installed.

WebSphere MQ bridge for HTTP runs on any JEE 1.4 compliant application server, by installing the WebSphere MQ resource adapter. You can also run WebSphere MQ bridge for HTTP on a WebSphere Application Server release earlier than version 6.0.2.1. Use the WebSphere Application Server Message Listener Port (MLP) to integrate WebSphere MQ as the JMS provider.

Support for WebSphere MQ bridge for HTTP is provided only for the following application servers:

- WebSphere Application Server 6.0.2.1 and later.
- WebSphere Application Server Community Edition Version 1.1 and later.

### About this task

WebSphere MQ bridge for HTTP is supplied as a `.war` file, `WMQHTTP.war`.

- On platforms other than z/OS®,
  - `WMQHTTP.war` is included as part of the "Java Messaging and Web Services" install option. The option is available in both the client and server installation materials.
  - `WMQHTTP.war` is installed to `<mqmtp>/java/http`. `<mqmtp>` is the directory where WebSphere MQ is installed.
- On z/OS,
  - `WMQHTTP.war` is included as part of the WebSphere MQ z/OS UNIX System Services Components feature.
  - `WMQHTTP.war` is installed to `PathPrefix/usr/lpp/mqm/V7R0M0/HTTPBridge/`, where `PathPrefix` is an optional customer defined prefix.

Carry out the following installation steps to install WebSphere MQ bridge for HTTP, deploy, and configure it, and verify the configuration. The details of the configuration steps vary on different application servers. Use [Deploying and verifying WebSphere MQ bridge for HTTP on WebSphere Application Server V6.1.0.9](#) as a template for the steps to follow on your application server.

### Procedure

1. Obtain `WMQHTTP.war` by installing either the WebSphere MQ client or server.
2. Copy `WMQHTTP.war` to a server from which it can be deployed to an application server.

3. Deploy `WMQHTTP.war` to an application server.
4. If necessary, install WebSphere MQ as a resource adapter on your application server. Find out if WebSphere MQ is already configured as a messaging provider on your application server. Use the administration or management tool supplied with your application server, to look for WebSphere MQ. WebSphere MQ might be found under the following path, **Resources > JMS > Messaging providers**.
5. Configure a connection factory on the application server to connect to a queue manager using the WebSphere MQ client transport<sup>1</sup>.
6. Configure the `WMQHTTP.war` Web application on the application server to use the connection factory
7. Verify the configuration.
  - a. Set up the queue manager named in the connection factory and a local queue.
  - b. Place a message on the local queue.
  - c. Create the server-connection channel named in the connection factory, with authority to read and write to the local queue.
  - d. Start the queue manager and the listener.
  - e. Start the application server and `WMQHTTP.war`, if they are not already running.
  - f. Open a browser and type `http://hostname:web_port/Context root/msg/queue/local queue`

## Results

The browser window displays the message you placed on the local queue.

## What to do next

1. Try the example, [Deploying and verifying WebSphere MQ bridge for HTTP on WebSphere Application Server V6.1.0.9](#).
2. Run the sample HTTP Java applications.

### [Deploying and verifying WebSphere MQ bridge for HTTP on WebSphere Application Server V6.1.0.9](#)

Use the following example to prepare a deployment of WebSphere MQ bridge for HTTP to run the sample HTTP Java programs. The deployment is on WebSphere Application Server V6.1.0.9.

### [Configuring WebSphere MQ Bridge for HTTP](#)

After installing the WebSphere MQ Bridge for HTTP and deploying `WMQHTTP.war` to your application server, configure the WebSphere MQ Bridge for HTTP so that it can implement diagnostic tracing and use your connection factory.

**Parent topic:** [WebSphere MQ bridge for HTTP](#)

## Related tasks

[Running the WebSphere MQ bridge for HTTP samples](#)


## Related information

[The WebSphere MQ resource adapter](#)

[WebSphere MQ Provider connection factory settings for application clients, V6.1](#)

[WebSphere MQ messaging provider connection factory settings, V7.0](#)

<sup>1</sup> Initially, at least, configure the client transport. Some application servers can connect to WebSphere MQ using direct, or bindings mode connections.

 This build: January 26, 2011 11:10:09

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts21160\_

## 2.2.1. Deploying and verifying WebSphere MQ bridge for HTTP on WebSphere Application Server V6.1.0.9

Use the following example to prepare a deployment of WebSphere® MQ bridge for HTTP to run the sample HTTP Java programs. The deployment is on WebSphere Application Server V6.1.0.9.

### Before you begin

1. Follow the instructions in [Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#), to copy `WMQHTTP.war` onto a server accessible to your installation of WebSphere Application Server.
2. Configure a queue manager, and a queue, to use to test the configuration:
  - o In the example, the queue manager is configured as using the values in [Table 1](#):

*Table 1. Queue manager configuration*

Object	Value
Host name	itso-01
Queue manager	QM1
Local queue	HTTPTESTQ
Server connection channel	MYSVRCON. Configure an MCA user ID with sufficient authority to read and write to HTTPTESTQ.
Listener port	1414

3. Start the queue manager and the listener
4. Place a test message onto `HTTPTESTQ`. For example:
  - a. Start WebSphere MQ Explorer.
  - b. In the list of local queues for QM1, right-click `HTTPTESTQ` > **Put test message** > **type First Message** > **Put message** > **Close**
5. Start the application server and sign on to the Integrated Solutions Console.



## About this task

The example shows the steps to take if you are running WebSphere Application Server V6.1.0.9 as your application server. If you are running a different version of WebSphere Application Server, or running a different application server, the steps are different. WebSphere Application Server V6.1.0.9 is pre-configured with WebSphere MQ installed as a message provider, using the WebSphere MQ client libraries. If WebSphere MQ is not pre-configured as a messaging provider, or if you want to use WebSphere MQ server bindings, you need to install and configure the WebSphere MQ resource adapter for JEE into your application server.

Follow the instructions to deploy WebSphere MQ bridge for HTTP onto WebSphere Application Server V6.1.0.9, and verify the deployment using a browser:

## Procedure

1. In the navigation pane, click **Resources > JMS providers > WebSphere MQ messaging provider**.  
You can configure at either Node, Cell, or Server level, depending on your WebSphere Application Server deployment. The example uses Server level deployment.
2. Under Additional properties, click **Connection factories > New**.
3. In the JMS providers form, provide the information in [Table 2](#), or alternatives of your choosing, click **Apply > Save**.

*Table 2. Set or modify the following fields*

Field	Value
Name	WMQHTTPBridge
JNDI Name	jms/WMQHTTPJCAConnectionFactory
Queue manager	QM1
Host	itso-01
Port	1414
Channel	MYSVRCON
Transport type	CLIENT

4. In the navigation pane, click **Applications > Install New Application**.
5. Insert the path to `WMQHTTP.war` into the form, and provide a Context root, click **Next**.
  - a. The Context root is optional. `mq` is the default Context root for the sample HTTP applications.
  - b. The Context root forms part of the URI identifying WebSphere MQ bridge for HTTP. You can omit the Context root, or change it later.
6. On the Select installation options page of the installation wizard, you do not have to change any of the defaults, click **Next**.
7. On the Map modules to servers page, select a Cluster or Server, check the Select box, click **Apply > Next**.
8. On the Map resource references to resources page, in the `javax.jms.ConnectionFactory` form, click **Browse...** on the IBM® WebSphere MQ bridge for HTTP row.
9. On the Enterprise Applications > Available resources page, select **WMQHTTPBridge**, click **Apply**.
10. Back in the `javax.jms.ConnectionFactory` form, select the authentication method.
  - a. For the example, choose **None**, click **Apply**. The other options require additional configuration.
11. Check the **Select** check box for IBM WebSphere MQ bridge for HTTP, click **Next > Next > Finish > Save**
12. In the navigation pane, click **Applications > Enterprise Applications**.
13. Check the selection box for `WMQHTTP.war`, click **Start**.
14. Open a browser window. Type `http://itso-01:9080/mq/msg/queue/HTTPTESTQ`, using the appropriate host name and port.


## Results

The browser window displays `First Message`, if the configuration is successful.

## What to do next

Run the sample HTTP Java applications.

**Parent topic:** [Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts21162\_



## 2.2.2. Configuring WebSphere MQ Bridge for HTTP

After installing the WebSphere® MQ Bridge for HTTP and deploying `WMQHTTP.war` to your application server, configure the WebSphere MQ Bridge for HTTP so that it can implement diagnostic tracing and use your connection factory.

The WebSphere MQ Bridge for HTTP has two sets of properties:

- Properties associated with diagnostic tracing. These properties are described in [Configuring WebSphere MQ Bridge for HTTP to implement diagnostic tracing](#).
- Properties associated with the resource adapter connection factory reference. How to configure these properties is described in [Configuring your connection factory](#).

### [Configuring WebSphere MQ Bridge for HTTP to implement diagnostic tracing](#)

A number of properties exist which are associated with diagnostic tracing in the WebSphere MQ Bridge for HTTP.

### [Configuring your connection factory](#)

Enable WebSphere MQ Bridge for HTTP to communicate with WebSphere MQ by configuring a connection factory. The procedure varies depending on the application server you are using.

**Parent topic:** [Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#)

This build: January 26, 2011 11:09:35

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts11420\_



### 2.2.2.1. Configuring WebSphere MQ Bridge for HTTP to implement diagnostic tracing

A number of properties exist which are associated with diagnostic tracing in the WebSphere MQ Bridge for HTTP.

[Table 1](#) lists the properties of the WebSphere MQ Bridge for HTTP that are associated with diagnostic tracing.



*Table 1. Properties of the WebSphere MQ Bridge for HTTP that are associated with diagnostic tracing*

Name of property	Type	Default value	Description
traceEnabled	String	false	A flag to enable or disable diagnostic tracing. If the value is false, tracing is turned off. If the value is true, a trace is sent to the location specified by the traceDestination property.
traceDestination	String	System.err	The location to where a diagnostic trace is sent. If the value is System.out, the trace is directed to the system output stream.
traceLevel	String	WARNINGS	The level of detail in a diagnostic trace. The value can be in the range NONE, which produces no trace, to ALL, which provides the most detail. See <a href="#">Table 2</a> for a description of each level.



[Table 2](#) describes the levels of detail for diagnostic tracing.



*Table 2. The levels of detail for diagnostic tracing*

Level	Level of detail
NONE	No trace.
EXCEPTIONS	The trace contains error messages.
WARNINGS	The trace contains error and warning messages.
INFO	The trace contains error, warning, and information messages.
ENTRYEXIT	The trace contains error, warning, and information messages, and entry and exit information for methods.
DATA	The trace contains error, warning, and information messages, entry and exit information for methods, and diagnostic data.
ALL	The trace contains all trace information.

**Note:** Specifying a trace level includes all trace statements at that level and above. For example, specifying a trace level of WARNINGS will cause all WARNINGS and EXCEPTIONS to be traced.



►If diagnostic tracing is turned off, error and warning messages are written to the system error stream. If diagnostic tracing is turned on, error messages are written to the system error stream and to the trace destination, but warning messages are written only to the trace destination. However, the trace contains warning messages only if the trace level is WARNINGS or higher. ◀

**Parent topic:** [Configuring WebSphere MQ Bridge for HTTP](#)

This build: January 26, 2011 11:09:35

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts11430\_



### 2.2.2.2. Configuring your connection factory

Enable WebSphere MQ Bridge for HTTP to communicate with WebSphere MQ by configuring a connection factory. The procedure varies depending on the application server you are using.

#### Before you begin

You have installed the WebSphere® MQ Bridge for HTTP and deployed WMQHTTP.war to your application server.

#### About this task

To enable the WebSphere MQ Bridge for HTTP to communicate with WebSphere MQ, specify the connection factory you want to use and configure that connection factory.

#### Procedure

- Optional: If you are using WebSphere Application Server Version 6 or later, complete the following step:
  - Set the connection factory `Broker version` property to `Basic` to use **Only character wildcards** for consistency with applications that previously used Broker Version 1, or set it to `Advanced` (the default) to use **Only topic wildcards** which are used in Broker Version 2. For more information about the **Broker version** property, see your WebSphere Application Server documentation
- Optional: If you are using the WebSphere MQ resource adapter as your WebSphere MQ JMS provider with WebSphere Application Server Community Edition, complete the following steps:
  - Locate and open your deployment plan. The deployment plan you use to deploy your resource adapter will be specific to the application

server you are using.

- b. Define a resource called `jms/WMQHTTPJCAConnectionFactory` with a value of the name of your `ConnectionFactory` object. Ensure that your `ConnectionFactory` is configured with the name of the WebSphere MQ queue manager that you want to connect to, if a user ID is required to access that queue manager ensure that this is configured on your `ConnectionFactory`.

### What to do next

If you are using WebSphere Application Server MLP with WebSphere Application Server, complete the following step:

1. Using the WebSphere Application Server admin console, create a `ConnectionFactory` object called `jms/WMQHTTPJCAConnectionFactory`. For information about how to do this refer to the WebSphere Application Server information center, [http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/tmm\\_ep.html](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/tmm_ep.html).

If a connection factory called `jms/WMQHTTPJCAConnectionFactory` does not exist when the Servlet is invoked for the first time (when it receives the first HTTP request), an `MQHTTP00002` error will occur and will be logged in your application server error log.

To confirm that the WebSphere MQ Bridge for HTTP is installed and configured correctly, in a web-browser navigate to `http://hostname:port/context_root/msg/queue/myqueue`, where `myqueue` is an empty WebSphere MQ queue. HTTP error 504 'Message retrieval timed out' will be displayed in the browser window if the WebSphere MQ Bridge for HTTP and JMS resource adapter are configured correctly.

**Parent topic:** [Configuring WebSphere MQ Bridge for HTTP](#)

This build: January 26, 2011 11:09:35

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts11170\_

## 2.3. Publish/subscribe using the WebSphere MQ bridge for HTTP

WebSphere® MQ bridge for HTTP uses the WebSphere MQ classes for JMS publish/subscribe interface. HTTP **POST** creates a publication. HTTP **DELETE** creates a non-durable managed subscription. You must configure publish/subscribe for JMS before using the topic URI.

Publish/subscribe is fully integrated into WebSphere MQ in version 7. Before version 7, a separate publish/subscribe broker handled publications and subscriptions. It is called "queued" publish/subscribe, to distinguish it from the fully integrated publish/subscribe in version 7. Version 7 emulates queued publish/subscribe using integrated publish/subscribe. The emulation enables existing queued publish/subscribe applications to coexist with integrated applications running on the same queue manager. Queued publish/subscribe applications can also interoperate with integrated applications, sharing the same topics. In version 6, the broker was shipped with WebSphere MQ; before version 6 it was available as a SupportPac.

### Configuration

The WebSphere MQ bridge for HTTP uses the JMS interface to publish and subscribe. In version 7, you can control whether the WebSphere MQ classes for JMS use queued or integrated publish/subscribe, using the `PROVIDERVERSION` JMS property.

An additional consideration is that you can use either WebSphere MQ client libraries with WebSphere MQ bridge for HTTP, or server libraries. Version 6 client libraries only support queued publish/subscribe, whereas version 7 libraries support both queued and integrated publish/subscribe. Most Web or application servers that use WebSphere MQ as a messaging provider do so using client libraries. In order to use integrated publish/subscribe, both the WebSphere MQ client and server libraries must be at least at version 7. If either is running an earlier version of WebSphere than 7, then you must configure queued publish/subscribe; see [Table 1](#). Check what libraries are installed or configured with the Web server or application server you are using.

Table 1. Publish/subscribe configuration modes

	Client V6 or earlier	Client V7 or later
<b>Server V6 or earlier</b>	<ol style="list-style-type: none"> <li>1. Run the <code>\java\bin\MQJMS_PSQ.mqsc</code> script</li> </ol>	Not supported
<b>Server V7 or later</b>	<ol style="list-style-type: none"> <li>1. Run the <code>\java\bin\MQJMS_PSQ.mqsc</code> script</li> <li>2. Set the queue manager to <code>PSMODE=ENABLED</code></li> </ol>	<ol style="list-style-type: none"> <li>1. If <code>PROVIDERVERSION = 7</code> <ol style="list-style-type: none"> <li>a. Set the queue manager to <code>PSMODE=ENABLED</code> or <code>PSMODE=COMPAT</code></li> </ol> </li> <li>2. If <code>PROVIDERVERSION = 6</code> <ol style="list-style-type: none"> <li>a. Set the queue manager to <code>PSMODE=ENABLED</code></li> </ol> </li> </ol>

### Publish

Send an HTTP **POST** request with the URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

The message contents are published using the topic string `topicString`.

### Subscribe

Send an HTTP **DELETE** request with the URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

WebSphere MQ bridge for HTTP creates a managed non-durable subscription to the topic string `topicString`. The subscription is deleted as soon as a publication is returned, or until the wait-interval set by the custom entity-header, `x-msg-wait`, expires.

**Parent topic:** [WebSphere MQ bridge for HTTP](#)

### Related information

[Properties of WebSphere MQ classes for JMS objects](#)

[The WebSphere MQ classes for JMS configuration file](#)

[When to use PROVIDERVERSION](#)

This build: January 26, 2011 11:10:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts21166\_

## 2.4. Running the WebSphere MQ bridge for HTTP samples

The WebSphere® MQ bridge for HTTP samples are available for use on only the Windows operating system. The samples show you how to submit HTTP **POST** and HTTP **DELETE** commands to WebSphere MQ bridge for HTTP from Java programs.

### Before you begin

Verify your WebSphere MQ bridge for HTTP installation by running step 7 in [Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#). The HTTP samples are installed to the directories shown in [Table 1](#). In each case, source code is installed to the `/src` subdirectory.

Table 1. Location of HTTP samples

Platform	Location
Windows	<code>mqmtop/tools/http/samples</code>
z/OS	<code>PathPrefix/usr/lpp/mqm/V7R0M0/http/samples</code>
i5/OS	<code>mqmtop/java/samples/http</code>
All other platforms	<code>mqmtop/samp/http</code>

### About this task

The samples simulate the WebSphere MQ AMQSPUT and AMQSGET sample applications. They illustrate the following functions in a point-to-point messaging environment:

- **HTTPPOST** - Sends HTTP **POST** requests in a Java application to put messages to a WebSphere MQ queue, using the WebSphere MQ bridge for HTTP and handles the responses.
- **HTTPDELETE** - Sends HTTP **DELETE** requests in a Java application to get messages from a WebSphere MQ queue, using the WebSphere MQ bridge for HTTP and handles the responses containing the WebSphere MQ message.

Parameters for HTTPPOST and HTTPDELETE

```

.-SYSTEM.DEFAULT.LOCAL.QUEUE-.  .-localhost:8080-.
>>+-----+-----+-----+----->
' -queueName-----' ' -hostName-----'

.-mq-----
>+-----+-----+-----+-----<
' -contextRoot-'

```

To run the **HTTPPOST** sample, complete the following steps:

### Procedure

1. In a command window, navigate to the HTTP samples directory.
2. Run the **HTTPPOST** sample.

```
java -classpath . HTTPPOST [parameters]
```

When the **HTTPPOST** sample starts, the following output is displayed:

```

HTTP POST Sample start
Target server is 'hostName'
Target queue is 'queueName'
Target context-root is 'contextRoot'

```

3. In the command prompt, type the text that you want to form your message body.
4. Press enter to post the message to the WebSphere MQ queue.
  - a. If you want to send another message, enter some more text. The texts forms the body of a second WebSphere MQ message.
  - b. Press enter to post the message to the WebSphere MQ queue.
5. Press enter twice to end **HTTPPOST**. The following output is displayed:

```
HTTP POST Sample end
```

### What to do next

The **HTTPDELETE** sample performs a destructive get of all the messages you placed on the WebSphere MQ queue.

Run the **HTTPDELETE** sample by completing the following steps:

1. In a command window, navigate to `WMQ InstallDir/tools/samples`.
2. Run the **HTTPDELETE** sample.

```
java -classpath . HTTPPOST [parameters]
```

When the **HTTPDELETE** sample starts, the following output is displayed:

```

HTTP DELETE Sample start
Target server is 'host:port'
Target queue is 'your queue name'
Target context-root is 'your context-root'
message
message
...

```

Parent topic: [WebSphere MQ bridge for HTTP](#)

### Related tasks

[Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#)

This build: January 26, 2011 11:10:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts11310\_

## 2.5. Security considerations for WebSphere bridge for HTTP

Standard Web security considerations apply to authenticating a Web browser client. Authorization to WebSphere® MQ resources is at the level of the user running the WebSphere Bridge for HTTP servlet, and not the individual Web browser client. Standard WebSphere MQ security consideration apply to WebSphere MQ.

Data flowing from a Web browser to a WebSphere MQ application using WebSphere bridge for HTTP, and back, takes a three steps:

### Client connection

From the browser to the WebSphere Bridge for HTTP over a TCP/IP connection using HTTP.

### Resource adapter connection to WebSphere MQ

The connection is from the WebSphere Bridge for HTTP to a WebSphere MQ queue manager. The connection is either a client connection, over TCP/IP, or a local WebSphere MQ bindings connection. Once the connection is made, the HTTP request is placed on a standard local queue or a transmission queue.

### From the WebSphere MQ local queue over one or more channels, to the target queue.

Apply standard techniques for securing queues, topics, queue managers, and channels.

The reply takes the steps in reverse.

### Client connection

Secure connections between HTTP clients and the application server using the Web container. Use standard HTTP server techniques, such as using HTTPS. Refer to the documentation for your application server for information.


### Resource adapter connection to WebSphere MQ

The connection between the resource adapter and queue manager is authorized using only a single user ID. Assign a single user ID to identify requests from the WebSphere Bridge for HTTP. The user ID must have restricted WebSphere MQ authorizations only to the resources external users must have access. You must authenticate the actual client separately, and establish trust for successive interactions with the client, using standard techniques for Web security.

Secure the connection between the resource adapter and the queue manager using the single user ID. Restrict the authorities the user ID has to no more than needed to read and write messages to queues and topics. The WebSphere Bridge for HTTP is a point of attack between the internet and your intranet.

How you secure the connection between your resource adapter and WebSphere MQ is dependent on your specific resource adapter. Refer to the documentation for the resource adapter.

**Parent topic:** [WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:08

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21150\_

## 2.6. Reference material for WebSphere MQ bridge for HTTP

Reference topics for WebSphere® MQ bridge for HTTP, arranged alphabetically

### [HTTP DELETE: WebSphere MQ bridge for HTTP command](#)

The HTTP **DELETE** operation gets a message from a WebSphere MQ queue, or retrieves a publication from a topic. The message is removed from the queue. If the publication is retained, it is not removed. A response message is sent back to the client including information about the message.

### [HTTP GET: WebSphere MQ bridge for HTTP command](#)

The HTTP **GET** operation gets a message from a WebSphere MQ queue. The message is left on the queue. The HTTP **GET** operation is equivalent to browsing a WebSphere MQ queue.

### [HTTP POST: WebSphere MQ bridge for HTTP command](#)

The HTTP **POST** operation puts a message on a WebSphere MQ queue, or publishes a message to a topic.

### [HTTP headers](#)

The WebSphere MQ bridge for HTTP supports custom request and entity HTTP headers, and a subset of standard HTTP headers.

### [HTTP return codes](#)

List of return codes from the WebSphere MQ bridge for HTTP


### [Message types and message mappings for WebSphere Bridge for HTTP](#)

WebSphere MQ bridge for HTTP supports four message classes, `TEXT`, `BYTES`, `STREAM` and `MAP`. The message classes are mapped to JMS message types and HTTP `Content-Type`.

### [URI Format](#)

URIs intercepted by WebSphere MQ bridge for HTTP.

**Parent topic:** [WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:18

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts22001\_

## 2.6.1. HTTP DELETE: WebSphere MQ bridge for HTTP command

The HTTP **DELETE** operation gets a message from a WebSphere® MQ queue, or retrieves a publication from a topic. The message is removed from the queue. If the publication is retained, it is not removed. A response message is sent back to the client including information about the message.

### Syntax

```
Request

>>+-DELETE+--- --| Path |-- --HTTP version--CRLF----->
' -GET-----'

      .-CRLF----- .-CRLF-----
      V             | V             |
>-----+-----+-----+-----+----->
' -general-header-' ' -request-header-'

      .-CRLF-----
      V             |
>-----+-----+-----<
' -| Entity-header (Request) |- '

Path

|--/--contextRoot--/----->

>--msg/--+queue/--queueName--+-----+-----|
|                                     ' -@--qMgrName-' |
' -topic/--topicName-----'

entity-header (Request)

|--+-----+-----|
+standard entity-header-- --entity-value--+
+x-msg-correlId - --correlation ID-----+
+x-msg-msgId - --message ID-----+
+x-msg-range-- --range-----+
+x-msg-require-headers-- --entity header name--+
'-x-msg-wait - --wait time-----'
```

### Note:

1. If a question mark (?) is used it must be substituted with %3f. For example, orange?topic should be specified as orange%3ftopic.
2. @qMgrName is only valid on an HTTP **POST**

```
Response

>>--HTTP version-- --HTTP Status-Code-- --HTTP Reason-Phrase--CRLF-->

      .-CRLF----- .-CRLF-----
      V             | V             |
>-----+-----+-----+----->
' -general-header-' ' -response-header-'

      .-CRLF-----
      V             |
>-----+-----+-----<
' -| Entity-header (Response) |- ' ' -CRLF--Message-'

entity-header (Response)

|--+-----+-----|
+standard entity-header-- --entity-value--+
+x-msg-class-- --message type-----+
+x-msg-correlId-- --correlation ID-----+
+x-msg-encoding-- --encoding type-----+
+x-msg-expiry-- --duration-----+
+x-msg-format-- --message format-----+
+x-msg-msgId-- --message ID-----+
+x-msg-persistence-- --persistence-----+
+x-msg-priority-- --priority class-----+
+x-msg-replyTo-- --reply-to queue-----+
+x-msg-timestamp-- --HTTP-date-----+
'-x-msg-usr-- --user properties-----'
```

### Request parameters

#### Path

See [URI Format](#).

#### HTTP version

HTTP version; for example, HTTP/1.1

#### general-header

See [HTTP/1.1 - 4.5 General Header Fields](#).

#### request-header

See [HTTP/1.1 - 5.3 Request Header Fields](#). The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

#### **entity-header (Request)**

See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity headers listed in the Request syntax diagram.

### Response parameters

#### **Path**

See [URI Format](#).

#### **HTTP version**

HTTP version; for example, HTTP/1.1

#### **general-header**

See [HTTP/1.1 - 4.5 General Header Fields](#).

#### **response-header**

See [HTTP/1.1 - 6.2 Response Header Fields](#).

#### **entity-header (Response)**

See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity or response headers listed in the Response syntax diagram. The Content-Length is always present in a response. It is set to zero if there is no message body.

#### **Message**

Message body.

### Description

If the HTTP **DELETE** request is successful, the response message contains the data retrieved from the WebSphere MQ queue. The number of bytes in the body of the message is returned in the HTTP Content-Length header. The status code for the HTTP response is set to 200 OK. If x-msg-range is specified as 0, or 0-0, then the status code of the HTTP response is 204 No Content.

If the HTTP **DELETE** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code.

### HTTP DELETE example

HTTP **DELETE** gets a message from a queue and deletes the message, or retrieves and deletes a publication. The **HTTPDELETE** Java sample is an example an HTTP **DELETE** request reading a message from a queue. Instead of using Java, you could create an HTTP **DELETE** request using a browser form, or an AJAX toolkit instead.

[Figure 1](#) is an HTTP request to delete the next message on queue called myQueue. In response, the message body is returned to the client. In WebSphere MQ terms, HTTP **DELETE** is a destructive get.

The request contains the HTTP request header x-msg-wait, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the x-msg-require-headers request header, which specifies that the client is to receive the message correlation ID in the response.

*Figure 1. Example of an HTTP **DELETE** request*

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

[Figure 2](#), is the response returned to the client. The correlation ID is returned to the client, as requested in x-msg-require-headers of the request.

*Figure 2. Example of an HTTP **DELETE** response*

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here's my message body that will appear on the queue.
```

**Parent topic:** [Reference material for WebSphere MQ bridge for HTTP](#)

#### **Related reference**

[correlId: HTTP x-msg-correlId entity-header](#)  
[msgId: HTTP x-msg-msgId entity-header](#)  
[range: HTTP x-msg-range request-header](#)  
[require-headers: HTTP x-msg-require-headers request-header](#)  
[wait: HTTP x-msg-wait request-header](#)  
[HTTP return codes](#)  
[URI Format](#)

#### **Related information**

[Hypertext Transfer Protocol -- HTTP/1.1](#)  
[HTTP/1.1 - 4.5 General Header Fields](#)  
[HTTP/1.1 - 5.3 Request Header Fields](#)  
[HTTP/1.1 - 6.2 Response Header Fields](#)  
[HTTP/1.1 - 7.1 Entity Header Fields](#)

 This build: January 26, 2011 11:10:12

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:

ts21240\_

## 2.6.2. HTTP GET: WebSphere MQ bridge for HTTP command

The HTTP **GET** operation gets a message from a WebSphere® MQ queue. The message is left on the queue. The HTTP **GET** operation is equivalent to browsing a WebSphere MQ queue.

### Syntax

```
Request

>>+-DELETE+--- --| Path |-- --HTTP version--CRLF----->
' -GET-----'

      .-CRLF----- .-CRLF-----
      V               | V               |
>-----+-----+-----+-----+----->
' -general-header-'   '-request-header-'

      .-CRLF----- .
      V               |
>-----+-----+-----<
' -| Entity-header (Request) |- '

Path

|--/--contextRoot--/------->

>--msg/--+--queue/--queueName--+-----+-----|
|                                     '-@--qMgrName-' |
' -topic/--topicName-----'

entity-header (Request)

|-----+-----|
+--standard entity-header-- --entity-value--+
+x-msg-correlId - --correlation ID-----+
+x-msg-msgId - --message ID-----+
+x-msg-range-- --range-----+
+x-msg-require-headers-- --entity header name--+
'-x-msg-wait - --wait time-----'
```

### Note:

1. If a question mark (?) is used it must be substituted with %3f. For example, orange?topic should be specified as orange%3ftopic.
2. @qMgrName is only valid on an HTTP **POST**

```
Response

>>--HTTP version-- --HTTP Status-Code-- --HTTP Reason-Phrase--CRLF-->

      .-CRLF----- .-CRLF-----
      V               | V               |
>-----+-----+-----+----->
' -general-header-'   '-response-header-'

      .-CRLF----- .
      V               |
>-----+-----+-----<
' -| Entity-header (Response) |- '   '-CRLF--Message-'

entity-header (Response)

|-----+-----|
+--standard entity-header-- --entity-value--+
+x-msg-class-- --message type-----+
+x-msg-correlId-- --correlation ID-----+
+x-msg-encoding-- --encoding type-----+
+x-msg-expiry-- --duration-----+
+x-msg-format-- --message format-----+
+x-msg-msgId-- --message ID-----+
+x-msg-persistence-- --persistence-----+
+x-msg-priority-- --priority class-----+
+x-msg-replyTo-- --reply-to queue-----+
+x-msg-timestamp-- --HTTP-date-----+
'-x-msg-usr-- --user properties-----'
```

### Request parameters

#### Path

See [URI Format](#).

#### HTTP version

HTTP version; for example, HTTP/1.1

#### general-header

See [HTTP/1.1 - 4.5 General Header Fields](#).

#### request-header

See [HTTP/1.1 - 5.3 Request Header Fields](#). The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

#### entity-header (Request)



See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity headers listed in the Request syntax diagram.

## Response parameters

### Path

See [URI Format](#).

### HTTP version

HTTP version; for example, HTTP/1.1

### general-header

See [HTTP/1.1 - 4.5 General Header Fields](#).

### response-header

See [HTTP/1.1 - 6.2 Response Header Fields](#).

### entity-header (Response)

See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

### Message

Message body.

## Description

If the HTTP **GET** request is successful, the response message contains the data retrieved from the WebSphere MQ queue. The number of bytes in the body of the message is returned in the HTTP `Content-Length` header. The status code for the HTTP response is set to 200 OK. If `x-msg-range` is specified as 0, or 0-0, then the status code of the HTTP response is 204 No Content.

If the HTTP **GET** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code.

## HTTP GET example

HTTP **GET** gets a message from a queue. The message remains on the queue. In WebSphere MQ terms, HTTP **GET** is a browse request. You could create an HTTP **DELETE** request using a Java client, a browser form, or an AJAX toolkit.

[Figure 1](#) is an HTTP request to browse the next message on queue called `myQueue`.

The request contains the HTTP request header `x-msg-wait`, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

*Figure 1. Example of an HTTP GET request*

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

[Figure 2](#) is the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

*Figure 2. Example of an HTTP GET response*

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here's my message body that will appear on the queue.
```


Parent topic: [Reference material for WebSphere MQ bridge for HTTP](#)

### Related reference

[correlId: HTTP x-msg-correlId entity-header](#)  
[msgId: HTTP x-msg-msgId entity-header](#)  
[range: HTTP x-msg-range request-header](#)  
[require-headers: HTTP x-msg-require-headers request-header](#)  
[wait: HTTP x-msg-wait request-header](#)  
[HTTP return codes](#)  
[URI Format](#)

### Related information

[Hypertext Transfer Protocol -- HTTP/1.1](#)  
[HTTP/1.1 - 4.5 General Header Fields](#)  
[HTTP/1.1 - 5.3 Request Header Fields](#)  
[HTTP/1.1 - 6.2 Response Header Fields](#)  
[HTTP/1.1 - 7.1 Entity Header Fields](#)

 This build: January 26, 2011 11:10:12

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts21230\_

## 2.6.3. HTTP POST: WebSphere MQ bridge for HTTP command

The HTTP **POST** operation puts a message on a WebSphere® MQ queue, or publishes a message to a topic.

## Syntax

```

Request

>>-POST-- --| Path |-- --HTTP version--CRLF----->

      .-CRLF----- .-CRLF-----
      V               | V               |
>-----+-----+-----+-----+----->
      '-general-header-'   '-request-header-'

      .-CRLF----- .-CRLF-----
      V               | V               |
>-----+-----+-----+-----+-----<
      '-| entity header (Request) |-'

Path

|--/--contextRoot--/------->

>--msg/--+--queue/--queueName--+/-----|
      |               '-@--qMgrName-' |
      '-topic/--topicName-----'

entity-header (Request)

|--+-----+-----|
+-standard entity-header-- --entity-value--+
+-x-msg-class-- --message type-----+
+-x-msg-correlId-- --correlation ID-----+
+-x-msg-encoding-- --encoding type-----+
+-x-msg-expiry-- --duration-----+
+-x-msg-format-- --message format-----+
+-x-msg-msgId-- --message ID-----+
+-x-msg-persistence-- --persistence-----+
+-x-msg-priority-- --priority class-----+
+-x-msg-replyTo-- --reply-to queue-----+
+-x-msg-require-headers-- --entity header name--+
'-x-msg-usr-- --user properties-----'

```

### Note:

1. If a question mark (?) is used it must be substituted with %3f. For example, orange?topic should be specified as orange%3ftopic.
2. @qMgrName is only valid on an HTTP **POST**

```

Response

>>-HTTP version-- --HTTP Status-Code-- --HTTP Reason-Phrase--CRLF-->

      .-CRLF----- .-CRLF-----
      V               | V               |
>-----+-----+-----+-----+----->
      '-general-header-'   '-response-header-'

      .-CRLF-----
      V               |
>-----+-----+-----+-----+-----<
      '-| entity-header (Response) |-'

entity-header (Response)

|--+-----+-----|
+-standard entity-header-- --entity-value--+
+-x-msg-class-- --message type-----+
+-x-msg-correlId-- --correlation ID-----+
+-x-msg-encoding-- --encoding type-----+
+-x-msg-expiry-- --duration-----+
+-x-msg-format-- --message format-----+
+-x-msg-msgId-- --message ID-----+
+-x-msg-persistence-- --persistence-----+
+-x-msg-priority-- --priority class-----+
+-x-msg-replyTo-- --reply-to queue-----+
+-x-msg-timestamp-- --HTTP-date-----+
'-x-msg-usr-- --user properties-----'

```

## Request parameters

### Path

See [URI Format](#).

### HTTP version

HTTP version; for example, HTTP/1.1

### general-header

See [HTTP/1.1 - 4.5 General Header Fields](#).

### request-header

See [HTTP/1.1 - 5.3 Request Header Fields](#). The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

### entity-header (Request)

See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity headers listed in the Request syntax diagram. The Content-Length and Content-Type

should be inserted in a request, and are often inserted automatically by the tool you use to create a client request. The `Content-Type` must match the type defined in the `x-msg-class` custom entity-header, if it is specified.

### Message

Message to put onto the queue, or publication to post to a topic.

### Response parameters

#### Path

See [URI Format](#).

#### HTTP version

HTTP version; for example, `HTTP/1.1`

#### general-header

See [HTTP/1.1 - 4.5 General Header Fields](#).

#### response-header

See [HTTP/1.1 - 6.2 Response Header Fields](#).

#### entity-header (Response)

See [HTTP/1.1 - 7.1 Entity Header Fields](#). One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

### Description

If no `x-msg-usr` header is included, and message class is `BYTES` or `TEXT`, the message put on the queue does not have an `MQRFH2`.

Use HTTP entity and request headers in the HTTP **POST** request to set the properties of the message that is put onto the queue. You can also use `x-msg-require-headers` to request which headers are returned in the response message.

If the HTTP **POST** request is successful, the entity of the response message is empty and its `Content-Length` is zero. The HTTP status code is `200 OK`.

If the HTTP **POST** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code. The WebSphere MQ message is not put on the queue or topic.

### HTTP POST example

HTTP **POST** puts a message to a queue, or a publication to a topic. The **HTTPPOST** Java sample is an example an HTTP **POST** request of a message to a queue. Instead of using Java, you could create an HTTP **POST** request using a browser form, or an AJAX toolkit instead.

[Figure 1](#) shows an HTTP request to put a message on a queue called `myQueue`. This request contains the HTTP header `x-msg-correlId` to set the correlation ID of the WebSphere MQ message.

Figure 1. Example of an HTTP **POST** request to a queue

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlId: 1234567890
Content-Length: 50

Here's my message body that will appear on the queue.
```

[Figure 2](#) shows the response sent back to the client. There is no response content.

Figure 2. Example of an HTTP **POST** response

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Parent topic: [Reference material for WebSphere MQ bridge for HTTP](#)

#### Related reference

[class: HTTP x-msg-class entity-header](#)  
[correlId: HTTP x-msg-correlId entity-header](#)  
[encoding: HTTP x-msg-encoding entity-header](#)  
[expiry: HTTP x-msg-expiry entity-header](#)  
[format: HTTP x-msg-format entity-header](#)  
[msgId: HTTP x-msg-msgId entity-header](#)  
[persistence: HTTP x-msg-persistence entity-header](#)  
[priority: HTTP x-msg-priority entity-header](#)  
[replyTo: HTTP x-msg-replyTo entity-header](#)  
[require-headers: HTTP x-msg-require-headers request-header](#)  
[usr: HTTP x-msg-usr entity-header](#)  
[HTTP return codes](#)  
[Message types and message mappings for WebSphere Bridge for HTTP](#)  
[URI Format](#)

#### Related information

[Hypertext Transfer Protocol -- HTTP/1.1](#)  
[HTTP/1.1 - 4.5 General Header Fields](#)  
[HTTP/1.1 - 5.3 Request Header Fields](#)  
[HTTP/1.1 - 6.2 Response Header Fields](#)  
[HTTP/1.1 - 7.1 Entity Header Fields](#)

This build: January 26, 2011 11:10:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21220\_

## 2.6.4. HTTP headers

The WebSphere® MQ bridge for HTTP supports custom request and entity HTTP headers, and a subset of standard HTTP headers.

HTTP practice is to prefix all custom headers with `x-`, the WebSphere MQ Bridge for HTTP headers are prefixed with `x-msg-`. For example, to set the priority header use `x-msg-priority`.

### Note:

- Most header values are case sensitive. For example, when using the `msgId` header, `NONE` is a keyword, whereas `none` is a `msgID`.
- Misspelled headers are ignored.

### Custom entity headers

The custom entity headers contain information about WebSphere MQ messages. Using entity headers, you can set values in the message descriptor (`MQMD`), or query values in the `MQMD`. An additional entity header, `x-msg-usr`, sets and returns any user property information you want to associate with a request.

You can use entity headers in different HTTP request contexts:

#### DELETE

You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **DELETE** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an `MQGET`, and to delete the message from its queue.

#### GET

You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **GET** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an `MQGET` for browse.

#### POST

You can use any entity header in a **POST** HTTP request, except `x-msg-timestamp`.

#### x-msg-require-headers

On any **GET**, **POST** or **DELETE** HTTP request, you can add multiple entity headers inside the `x-msg-require-headers` request header, separated by commas. The effect is to return the specified entity headers in the HTTP response message, containing the value of the associated message property.

The description of each header lists in which contexts the header is processed by WebSphere MQ bridge for HTTP. For example, in [Table 1](#), the header is processed by WebSphere MQ bridge for HTTP in an HTTP **POST** request, or in the `x-msg-require-headers` request header in either an HTTP **POST**, **GET**, or **DELETE** request. If the header is included in a context it is not allowed in, the header is ignored. No error is reported.

You can put any standard HTTP headers into requests to be processed by the Web server, or other request handlers. Similarly, the response might contain other standard HTTP headers inserted by the Web server or other response handlers.

Table 1. Example of how the allowed contexts is documented.

Valid in HTTP request message	<b>POST</b> , <code>x-msg-require-headers</code>
-------------------------------	--

### Custom request headers

The three custom request headers, `x-msg-range`, `x-msg-require-headers`, and `x-msg-wait`, pass additional information about the HTTP request to the server. They act as request modifiers. With `x-msg-range`, you can restrict the amount of message data returned in a response. With `x-msg-require-headers`, you can request the response to contain information about the result of the request. With `x-msg-wait`, you can modify the time the client waits for an HTTP response.

### Standard HTTP headers

The `Host` standard HTTP request-header must be specified in an HTTP/1.1 request.

The `Content-Length` and `Content-Type` standard HTTP entity headers can be specified in a request.

The `Content-Length`, `Content-Location`, `Content-Range`, `Content-Type`, and `Server` standard HTTP entity headers can be returned in response to a request. Specify one or more of the standard HTTP headers in the `x-msg-request-header` header in the request message.

### Alphabetic list of headers

**class: HTTP x-msg-class entity-header**  
Set or return the message type.

**Content-Length: HTTP entity-header**  
Set or return the length, in bytes, of the body of the message.

**Content-Location: HTTP entity-header**  
Returns the queue or topic referenced in the request, in the standard `Content-Location` header in the HTTP response message.

**Content-Range: HTTP entity-header**  
Return the range of bytes selected from a WebSphere MQ message in the `Content-Range` header in an HTTP response.

**Content-Type: HTTP entity-header**  
Set or return the class of the JMS message in a WebSphere MQ message according to the HTTP content-type.

**correlId: HTTP x-msg-correlId entity-header**  
Set or return the correlation identifier.

**encoding: HTTP x-msg-encoding entity-header**

Set or return the message encoding.

**expiry:** [HTTP x-msg-expiry entity-header](#)

Set or return the message expiry duration.

**format:** [HTTP x-msg-format entity-header](#)

Set or return the WebSphere MQ message format.

**msgId:** [HTTP x-msg-msgId entity-header](#)

Set or return the message identifier.

**persistence:** [HTTP x-msg-persistence entity-header](#)

Set or return the message persistence.

**priority:** [HTTP x-msg-priority entity-header](#)

Set or return the message priority.

**range:** [HTTP x-msg-range request-header](#)

Return a range of bytes from a message.

**replyTo:** [HTTP x-msg-replyTo entity-header](#)

Set or return the message reply-to queue and queue manager name.

**Server:** [HTTP response-header](#)

Returns information about the server and protocol the client is connected to.

**require-headers:** [HTTP x-msg-require-headers request-header](#)

Set which headers to return in the HTTP response message.

**timestamp:** [HTTP x-msg-timestamp entity-header](#)

Return the message time stamp.

**usr:** [HTTP x-msg-usr entity-header](#)

Set or return the user properties.


**wait:** [HTTP x-msg-wait request-header](#)

Set the period of time to wait for a message to arrive, before returning an HTTP 504 Gateway Timeout response message.

**Parent topic:** [Reference material for WebSphere MQ bridge for HTTP](#)

**Related information**

[Hypertext Transfer Protocol -- HTTP/1.1](#)

 This build: January 26, 2011 11:10:13

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts21250\_

## 2.6.4.1. class: HTTP x-msg-class entity-header

Set or return the message type.

Type	Description
HTTP header name	x-msg-class
HTTP header type	Entity-header
Valid in HTTP request message	POST, x-msg-require-headers
Allowed values	<b>BYTES</b> <b>MAP</b> <b>STREAM</b> <b>TEXT</b>
Default value	BYTES

### Description

- In an HTTP **POST** request, sets the type of the message created.
- Specifying the class header on a **GET** or **DELETE** returns a 400 Bad Request with entity body of MQHTTP40007.
- Specified in x-msg-require-headers, sets x-msg-class in the HTTP response message to the type of a message.
- If an invalid value is specified for this header a MQHTTP40005 message is returned.
- If the x-msg-class header is not specified and the content-type of the message is application/x-www-form-urlencoded, the data is assumed to be a JMS map object.

**Parent topic:** [HTTP headers](#)

 This build: January 26, 2011 11:10:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts22010\_

### 2.6.4.2. Content-Length: HTTP entity-header

Set or return the length, in bytes, of the body of the message.

Type	Description
HTTP header name	Content-Length
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Allowed and returned value	<b>Integer value</b> Length in bytes of the message body.


#### Description

- The `Content-Length` is optional in an HTTP request. For a **GET** or **DELETE** the length must be zero. For **POST**, if `Content-Length` is specified and it does not match the length of the message-line, the message is either truncated, or padded with nulls to the specified length.
- The `Content-Length` is always returned in the HTTP response even when there is no content, in which case the value is zero.

Parent topic: [HTTP headers](#)

#### Related information

[HTTP/1.1 - 14.13 Content-Length](#)

 This build: January 26, 2011 11:10:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22162\_

### 2.6.4.3. Content-Location: HTTP entity-header

Returns the queue or topic referenced in the request, in the standard `Content-Location` header in the HTTP response message.

Type	Description
HTTP header name	Content-Location
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Returned value	URI in the format,  <i>/msg/queue/queuename</i>  or  <i>/msg/topic/topicname</i>


#### Description

- When requested in `x-msg-require-headers`, the `Content-Location` entity-header returns the queue or topic referenced in the HTTP request.

Parent topic: [HTTP headers](#)

#### Related information

[HTTP/1.1 - 14.14 Content-Location](#)

 This build: January 26, 2011 11:10:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22150\_

### 2.6.4.4. Content-Range: HTTP entity-header

Return the range of bytes selected from a WebSphere MQ message in the `Content-Range` header in an HTTP response.

Type	Description
HTTP header name	Content-Range
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Returned value	<b>String</b> Returns the lower limit, <i>m</i> and upper limit, <i>n</i> of the returned substring, and <i>length</i> of the whole message. For example,  <i>m-n/length</i>

#### Description

- The `Content-Range` is only returned in the HTTP response when `Content-Range` is specified in a **GET** or **DELETE** request that contains an `x-msg-range` request header.
- If `x-msg-range` is specified on a **GET** or **DELETE** request, the range of bytes specified in the `Content-Range` header are returned in the response. For example, if `x-msg-range: 0-60` is used in a request for a message containing 100 bytes, the content-range header holds the string `0-60/100`

- An `x-msg-range` request also returns the content range in the `x-msg-range` header in the HTTP response.

Parent topic: [HTTP headers](#)

#### Related information

[HTTP/1.1 - 14.16 Content-Range](#)

This build: January 26, 2011 11:10:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22160\_

### 2.6.4.5. Content-Type: HTTP entity-header

Set or return the class of the JMS message in a WebSphere® MQ message according the to HTTP content-type.

Type	Description
HTTP header name	Content-Type
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , <code>x-msg-require-headers</code>
Allowed or returned value	<b>media-type</b> For media-types that are supported see <a href="#">Table 1</a> .

Table 1. Mapping between `x-msg-class` and HTTP `Content-Type`

<code>x-msg-class</code>	HTTP <code>Content-Type</code>
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

#### Description

- On an HTTP **POST** request, specify either the `Content-Type` or the `x-msg-class`. If you specify both, they must be consistent or an HTTP Bad Request exception, Status code 400 is returned. If you omit both, the `Content-Type` and the `x-msg-class`, a `Content-Type` of `text/*` is assumed.
- The `Content-Type` is always set in the response to an HTTP **GET** or **DELETE** that has a message body. The `Content-Type` is set according to the rules in [Table 2](#).

Table 2. Mapping message types to `x-msg-class` and `Content-Type`

Message format	JMS Message type	<code>x-msg-class</code>	<code>Content-Type</code>
Anything except MQFMT_STRING	None	BYTES	application/octet-stream
MQFMT_STRING	None	TEXT	text/plain
MQFMT_NONE	jms_bytes	BYTES	application/octet-stream
MQFMT_NONE	jms_text	TEXT	text/plain
MQFMT_NONE	jms_map	MAP	application/xml
MQFMT_NONE	jms_stream	STREAM	application/xml

Parent topic: [HTTP headers](#)

#### Related information

[HTTP/1.1 - 14.17 Content-Type](#)

[HTTP/1.1 - 3.7 media-type](#)

This build: January 26, 2011 11:10:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22160\_

### 2.6.4.6. correlId: HTTP x-msg-correlId entity-header

Set or return the correlation identifier.

Type	Description
HTTP header name	<code>x-msg-correlId</code>
HTTP header type	Entity-header
Valid in HTTP request message	<b>DELETE</b> , <b>GET</b> , <b>POST</b> , <code>x-msg-require-headers</code>
Allowed values	<b>String value</b> For example: <code>x-msg-correlId: mycorrelationid</code> Strings enclosed in double quotation marks are permitted; for example:

	<pre>x-msg-correlId: "my id"</pre> <p><b>Hex value</b></p> <p>A hex value prefixed with 0x:; for example:</p> <pre>x-msg-correlId: 0x:43c1d23a</pre> <p>►The hex value following 0x: is limited to 48 characters representing 24 bytes. Additional data is ignored.◀</p>
Default value	Not applicable

**Description**

- On an HTTP **POST** request, sets the correlation ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified correlation ID, an HTTP 504 Gateway Timeout response is returned. `x-msg-correlId` can be used with `x-msg-msgID` to select a message from a queue or topic that matches both selectors.
- Specified in `x-msg-require-headers`, sets `x-msg-coreId` in the HTTP response message to the correlation ID of a message.
- Horizontal white space is allowed after the `0x:` prefix.

**Note:**

- Specifying `x-msg-correlId` without a value on an HTTP **GET** or **DELETE** request; for example, "`x-msg-correlId:`", returns the next message on the queue or topic regardless of its correlation ID.
- ►If you specify a selector of 24 characters or fewer, or `0x:` followed by 48 characters or fewer, WebSphere® MQ bridge for HTTP uses an optimized selector for improved performance.◀
- A JMS message selector containing `JMSCorrelationID` is used when selecting messages from the queue. This selector behaves as described in [Selection Behavior](#).

Parent topic: [HTTP headers](#)

This build: January 26, 2011 11:10:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts22020\_

### 2.6.4.7. encoding: HTTP `x-msg-encoding` entity-header

Set or return the message encoding.

Type	Description
HTTP header name	<code>x-msg-encoding</code>
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , <code>x-msg-require-headers</code>
Allowed values	A comma-separated list of the following values:  <pre>DECIMAL_NORMAL DECIMAL_REVERSED FLOAT_IEEE_NORMAL FLOAT_IEEE_REVERSED FLOAT_S390 INTEGER_NORMAL INTEGER_REVERSED</pre> For example, <pre>x-msg-encoding: INTEGER_NORMAL,DECIMAL_NORMAL,FLOAT_IEEE_NORMAL</pre> <p><b>Note:</b> The value is case-sensitive</p>
Default value	<code>DECIMAL_NORMAL, FLOAT_IEEE_NORMAL, INTEGER_NORMAL</code>

**Description**

- On an HTTP **POST** request, specifies the encoding of the message created.
- On an HTTP **GET** or **DELETE** request, the `x-msg-encoding` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-encoding` in the HTTP response message to the encoding property of a message.

Parent topic: [HTTP headers](#)

This build: January 26, 2011 11:10:25

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
 This topic's URL:  
 ts22030\_

### 2.6.4.8. expiry: HTTP `x-msg-expiry` entity-header

Set or return the message expiry duration.

Type	Description
HTTP header name	<code>x-msg-expiry</code>




HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , x-msg-require-headers
Allowed values	<p><b>UNLIMITED</b></p> <p>For example;</p> <p style="padding-left: 40px;">x-msg-expiry: UNLIMITED</p> <p><b>Integer value</b></p> <p>Milliseconds before expiry.</p> <p>For example;</p> <p style="padding-left: 40px;">x-msg-expiry: 10000</p>
Default value	UNLIMITED

### Description

- When set on an HTTP **POST** request, the request message expires in the time specified.
- On an HTTP **GET** or **DELETE** request, the x-msg-expiry header is ignored.
- Specified in x-msg-require-headers, sets x-msg-expiry in the HTTP response message to the expiry time of a message.
- **UNLIMITED** specifies that the message never expires.
- The expiry of a message starts from the time the message arrives on the queue, as a result network latency is ignored.
- The maximum value is limited by WebSphere® MQ to 214748364700 milliseconds. If a value greater than this is specified then the maximum possible expiry time is assumed.

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:25

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts22040\_

## 2.6.4.9. format: HTTP x-msg-format entity-header

Set or return the WebSphere MQ message format.

Type	Description
HTTP header name	x-msg-format
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , x-msg-require-headers
Allowed values	<p><b>NONE</b></p> <p>For example,</p> <p style="padding-left: 40px;">x-msg-format: NONE</p> <p><b>String value</b></p> <p>Any user-defined value of up to eight characters. For example,</p> <p style="padding-left: 40px;">x-msg-format: myformat</p>
Default value	None


### Description

- When set on an HTTP **POST** request, set the request message format.
- On an HTTP **GET** or **DELETE** request, the x-msg-format header is ignored.
- Specified in x-msg-require-headers, sets x-msg-format in the HTTP response message to the format of a message.
- **NONE** is case sensitive, and indicates that the message format is blank.
- The value of x-msg-format is used, even if it contradicts the media-type of the HTTP request. See [Table 1](#).

Table 1. Mapping content-type and x-msg-class to message format

x-msg-class	Content-type	Message format on queue/topic
BYTES	<ul style="list-style-type: none"> <li>• application/octet-stream</li> <li>• application/xml</li> </ul>	WebSphere® MQ message: MQFMT set to MQC.MQFMT_NONE
TEXT	<ul style="list-style-type: none"> <li>• text/*</li> </ul>	WebSphere MQ message: MQFMT set to MQC.MQFMT_STRING
MAP	<ul style="list-style-type: none"> <li>• application/x-www-form-urlencoded</li> <li>• application/xml (optional)</li> </ul>	JMSMap
STREAM	<ul style="list-style-type: none"> <li>• application/xml (optional)</li> </ul>	JMSStream

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:25

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts22050\_

### 2.6.4.10. msgId: HTTP x-msg-msgId entity-header

Set or return the message identifier.

Type	Description
HTTP header name	x-msg-msgId
HTTP header type	Entity-header
Valid in HTTP request message	<b>DELETE, GET, POST</b> , x-msg-require-headers
Allowed values	<p><b>String value</b></p> <p>For example,</p> <pre>x-msg-msgId: mymsgid</pre> <p>Strings enclosed in quotation marks for example, x-msg-msgId: "my id"</p> <p><b>Hex value</b></p> <p>A hex value prefixed with 0x:; for example,</p> <pre>x-msg-msgId: 0x:43c1d23a</pre>
Default value	Not applicable


#### Description

- On an HTTP **POST** request, sets the message ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified message ID, an HTTP 504 Gateway Timeout response is returned. x-msg-msgId can be used with x-msg-correlID to select a message from a queue or topic that matches both selectors.
- Specified in x-msg-require-headers, returns x-msg-msgId in the HTTP response to the message ID of a message.
- Horizontal white space is allowed after the 0x: prefix.

**Note:** Specifying x-msg-msgId without a value on an HTTP **GET** or **DELETE** request; for example, "x-msg-msgId:", returns the next message on the queue or topic regardless of its message ID.

A JMS message selector containing JMSMessageID is used when selecting messages from the queue. This selector behaves as described in [Selection behavior](#).

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:25

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22060\_

### 2.6.4.11. persistence: HTTP x-msg-persistence entity-header


Set or return the message persistence.

Type	Description
HTTP header name	x-msg-persistence
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , x-msg-require-headers
Allowed values	<p><b>NON_PERSISTENT</b></p> <p>The message does not survive system failures or queue manager restarts.</p> <p>For example,</p> <pre>x-msg-persistence: NON_PERSISTENT</pre> <p><b>PERSISTENT</b></p> <p>The message survives system failures and restarts of the queue manager.</p> <p>For example,</p> <pre>x-msg-persistence: PERSISTENT</pre> <p><b>AS_DESTINATION</b></p> <p>Applies to <b>POST</b> only.</p> <p>Use the default persistence of the destination as determined by the message provider.</p> <p><b>Note:</b> Case sensitive</p>
Default value	NON_PERSISTENT

#### Description

- When set on an HTTP **POST** request, set the request message persistence.
- On an HTTP **GET** or **DELETE** request, the x-msg-persistence header is ignored.
- Specified in x-msg-require-headers, sets x-msg-persistence in the HTTP response message to the persistence of a message.

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:26

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22090\_

### 2.6.4.12. priority: HTTP x-msg-priority entity-header


Set or return the message priority.

Type	Description
HTTP header name	x-msg-priority
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , x-msg-require-headers
Allowed values	<p><b>LOW</b></p> <p>For example,</p> <p style="padding-left: 40px;">x-msg-priority: LOW</p> <p><b>MEDIUM</b></p> <p>This priority is equal to a WebSphere® MQ priority level of 4. For example,</p> <p style="padding-left: 40px;">x-msg-priority: MEDIUM</p> <p><b>HIGH</b></p> <p>For example,</p> <p style="padding-left: 40px;">x-msg-priority: HIGH</p> <p><b>Integer value</b></p> <p>A string representation of an integer between 0 - 9; for example,</p> <p style="padding-left: 40px;">x-msg-priority: 3</p> <p><b>AS_DESTINATION</b></p> <p>Applies to <b>POST</b> only. Use the default priority of the destination as determined by the message provider.</p> <p><b>Note:</b> Case sensitive</p>
Default value	MEDIUM

#### Description

- When set on an HTTP **POST** request, set the request message priority.
- On an HTTP **GET** or **DELETE** request, the x-msg-priority header is ignored.
- Specified in x-msg-require-headers, sets x-msg-priority in the HTTP response message to the priority of a message.

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:26

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22100\_

### 2.6.4.13. range: HTTP x-msg-range request-header


Return a range of bytes from a message.

Type	Description
HTTP header name	x-msg-range
HTTP header type	Request-header
Valid in HTTP request message	<b>GET</b> , <b>DELETE</b>
Allowed value	<p><b>Integer value <i>n</i></b></p> <p>Returns the first <i>n</i> bytes of the message.</p> <p>If <i>n</i> = 0, the result is an HTTP 204 - No Content response code.</p> <p><b>Integer values <i>n-m</i></b></p> <p style="padding-left: 40px;"><i>n</i> &lt; <i>m</i></p> <p>Returns a range of bytes from the message content, from <i>n</i> bytes to <i>m</i> bytes inclusive.</p> <p><b>ALL</b></p> <p>The whole of the message content is returned in the response message.</p>
Default value	Not applicable

#### Description

- On an HTTP **POST**, the x-msg-range header is ignored.
- On an HTTP **GET** or **DELETE**, x-msg-range specifies the range of bytes returned in the response message. The range of bytes is returned in the x-msg-range header in the response message. For example, if x-msg-range: 0-60 is used in a request for a message containing 100 bytes, the content-range header holds the string 0-60/100.
- If no data is requested, using x-msg-range: 0 or x-msg-range: 0-0, the result is an HTTP 204 - No Content" response.
- If an invalid range is specified, for example, if *n* is greater than *m*, or the syntax is incorrect, the result is an HTTP 400 Bad Request error with entity body MQHTTP40005.
- If x-msg-range is specified on anything but a **GET** or **DELETE** request, the result is an HTTP 400 Bad Request error with entity body MQHTTP40007.
- If a valid range is specified on a **GET** or **DELETE** request, the result is an HTTP 1.1 Content-Range header as specified in the HTTP 1.1 specification.

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22110\_

#### 2.6.4.14. replyTo: HTTP x-msg-replyTo entity-header

Set or return the message reply-to queue and queue manager name.

Type	Description
HTTP header name	x-msg-replyTo
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , x-msg-require-headers
Allowed values	<p><b>URI</b></p> <p>A point-to-point URI; for example,</p> <pre>x-msg-replyTo: /msg/queue/myReplyQueue x-msg-replyTo: /msg/queue/myReplyQueue@myReplyQueueManager</pre> <p><b>Note:</b> Case sensitive</p>
Default value	MEDIUM

##### Description


- When set on an HTTP **POST** request, set the request message replyTo destination.
- On an HTTP **GET** or **DELETE** request, the x-msg-replyTo header is ignored.
- Specified in x-msg-require-headers, sets x-msg-replyTo in the HTTP response message to the reply-to queue and queue manager name of a message.

**Note:** The URI in the HTTP response can include the name of the queue manager to which the WebSphere® MQ bridge for HTTP is connected.

**Parent topic:** [HTTP headers](#)

##### Related reference

[URI Format](#)

 This build: January 26, 2011 11:10:26

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22080\_

#### 2.6.4.15. Server: HTTP response-header

Returns information about the server and protocol the client is connected to.

Type	Description
HTTP header name	Server
HTTP header type	Response-header
Valid in HTTP request message	x-msg-require-headers
Returned value	<p>WMQ-HTTP/1.1 JEE-Bridge/1.1</p> <p>or</p> <p>Server: <i>Product-token</i> WMQ-HTTP/1.1 JEE-Bridge/1.1</p>

##### Description


- If WebSphere® MQ Bridge for HTTP is deployed to an application server, the WebSphere MQ bridge for HTTP details is appended to the server response header. For example, the WebSphere MQ bridge for HTTP deployed to WebSphere Application Server Community Edition, called Apache-Coyote, gives the response:

```
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
```

**Parent topic:** [HTTP headers](#)

##### Related information

[HTTP/1.1 - 14.38 Server](#)

 This build: January 26, 2011 11:10:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts22180\_

#### 2.6.4.16. require-headers: HTTP x-msg-require-headers request-header

Set which headers to return in the HTTP response message.

Type	Description
------	-------------

HTTP header name	x-msg-require-headers
HTTP header type	Request-header
Valid in HTTP request message	<b>POST, GET, DELETE</b>
Allowed values	<p>A comma-separated list of the entity header names:</p> <p><b>ALL</b></p> <p><b>ALL-USR</b></p> <p><b>class</b></p> <p><b>content-location</b></p> <p><b>correlId</b></p> <p><b>encoding</b></p> <p><b>expiry</b></p> <p><b>format</b></p> <p><b>msgId</b></p> <p><b>NO_require-headers</b></p> <p><b>persistence</b></p> <p><b>priority</b></p> <p><b>replyTo</b></p> <p><b>server</b></p> <p><b>timestamp</b></p> <p><b>usr-property name</b></p> <p>For example,</p> <p style="padding-left: 20px;">x-msg-require-headers: msgId</p> <p>Or,</p> <p style="padding-left: 20px;">x-msg-require-headers: expiry, correlId, timestamp</p> <p>To request a specific property:</p> <p style="padding-left: 20px;">x-msg-require-headers: usr-myCustomProperty</p> <p>To request all properties:</p> <p style="padding-left: 20px;">x-msg-require-headers: ALL-USR, ALL</p>
Default value	NO_require-headers


### Description

- The value of x-msg-require-headers is not case-sensitive, except in the cases of the ALL, NO\_require-headers, and ALL-USR constants, and the *property-name* variable.

Parent topic: [HTTP headers](#)

### Related reference

[class: HTTP x-msg-class entity-header](#)  
[Content-Location: HTTP entity-header](#)  
[Content-Range: HTTP entity-header](#)  
[correlId: HTTP x-msg-correlId entity-header](#)  
[encoding: HTTP x-msg-encoding entity-header](#)  
[expiry: HTTP x-msg-expiry entity-header](#)  
[format: HTTP x-msg-format entity-header](#)  
[msgId: HTTP x-msg-msgId entity-header](#)  
[persistence: HTTP x-msg-persistence entity-header](#)  
[priority: HTTP x-msg-priority entity-header](#)  
[replyTo: HTTP x-msg-replyTo entity-header](#)  
[Server: HTTP response-header](#)  
[timestamp: HTTP x-msg-timestamp entity-header](#)  
[usr: HTTP x-msg-usr entity-header](#)  
[URI Format](#)

 This build: January 26, 2011 11:10:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22140\_

## 2.6.4.17. timestamp: HTTP x-msg-timestamp entity-header

Return the message time stamp.

Type	Description
HTTP header name	x-msg-timestamp
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Returned value	<p><b>HTTP-date</b></p> <p>A date in the format; day, date month year time time-zone; for example,</p> <p style="padding-left: 20px;">Sun, 06 Nov 1994 08:49:37 GMT</p> <p>Defined by RFC 822, and updated in RFC 1123.</p>
Default value	Not applicable


**Description**

- On an HTTP **POST**, **GET** or **DELETE** request, the `x-msg-timestamp` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-timestamp` in the HTTP response message to the timestamp of a message.

Parent topic: [HTTP headers](#)

**Related information**

[HTTP/1.1 - 14.18 Date](#)

 This build: January 26, 2011 11:10:26

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22070\_

**2.6.4.18. usr: HTTP x-msg-usr entity-header**

Set or return the user properties.

Type	Description
HTTP header name	<code>x-msg-usr</code>
HTTP header type	Entity-header
Valid in HTTP request message	<b>POST</b> , <code>x-msg-require-headers</code>
Allowed values	See <a href="#">Syntax</a> ; for example, <code>x-msg-usr: myProp1;5;i1,</code> <code>x-msg-usr: myProp2;"My String";string</code>
Default value	Not applicable

**Description**

- When set on an HTTP **POST** request, set the request message user properties.
- On an HTTP **GET** or **DELETE** request, the `x-msg-usr` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-usr` in the HTTP response message to user properties of a message.
- Multiple properties can be set on a message. Specify multiple comma-separated properties in a single `x-msg-usr` header, or by using two or more separate instances of the `x-msg-usr` header.
- You can request a specific property to be returned in the response to a **GET** or **DELETE** request. Specify the name of the property in the `x-msg-require-headers` header of the request, using the prefix `usr-`. For example,  
`x-msg-require-headers: usr-myProp1`
- To request that all user properties are returned in a response, use the `ALL-USR` constant. For example,  
`x-msg-require-headers: ALL-USR`

**Syntax**


```

      .-,-----
      v |
>>-x-msg-usr:---| usr-property-value |-----<
usr-property-value
|--| property-name |--;--| usr-value |--;--| usr-type |-----|
property-name
|--string-----|
usr-value
|---+| boolean |-----|
+| i1 |-----+
+| i2 |-----+
+| i4 |-----+
+| i8 |-----+
+| r4 |-----+
+| r8 |-----+
'| qstring |-|
usr-type
|---boolean+-----|
+-i1-----+
+-i2-----+
+-i4-----+
+-i8-----+
+-r4-----+
+-r8-----+
'-string--|
boolean
|---TRUE---+-----|
'-FALSE-'
i1

```

```
|-- -128-- ≤ n ≤ --+127-----|
i2
|-- -32768-- ≤ n ≤ --+32767-----|
i4
|-- -2147483648-- ≤ n ≤ --+2147483647-----|
i8
|-- -9223372036854775808-- ≤ n ≤ --+92233720368547750807-----|
r4
|-- -1.4E-45-- ≤ n ≤ --+3.4028235E38-----|
r8
|-- -4.9E-324-- ≤ n ≤ --+1.7976931348623157E308-----|
qstring
|--"--string--"-----|
```

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22120\_

### 2.6.4.19. wait: HTTP x-msg-wait request-header


Set the period of time to wait for a message to arrive, before returning an HTTP 504 Gateway Timeout response message.

Type	Description
HTTP header name	x-msg-wait
HTTP header type	Request-header
Valid in HTTP request message	<b>GET, DELETE</b>
Allowed value	<p><b>NO_WAIT</b></p> <p>For example,</p> <p style="padding-left: 40px;">x-msg-wait: NO_WAIT</p> <p><b>Integer value</b></p> <p>The number of milliseconds that the WebSphere MQ bridge for HTTP waits for a message to arrive; for example,</p> <p style="padding-left: 40px;">x-msg-wait: 8</p>
Default value	NO_WAIT

#### Description

- On an HTTP **POST** request, the x-msg-wait header is ignored.
- On an HTTP **GET** or **DELETE** request, x-msg-wait specifies time to wait for a message to arrive before returning an HTTP 504 Gateway Timeout response.
- NO\_WAIT is case-sensitive.
- The default maximum wait time is 35000. You can change the default by setting the maximum\_wait\_time parameter of the servlet. See the [Installing, configuring, and verifying WebSphere MQ bridge for HTTP](#) section for more information.
- If you set a value greater than maximum\_wait\_time, maximum\_wait\_time is used instead.

Parent topic: [HTTP headers](#)

 This build: January 26, 2011 11:10:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts22130\_

### 2.6.5. HTTP return codes

List of return codes from the WebSphere® MQ bridge for HTTP

The WebSphere MQ bridge for HTTP returns four types of error:

#### Servlet errors

MQHTTP0001 and MQHTTP0002 are servlet errors. They are logged, but not returned to the HTTP client.

#### Successful operations

An HTTP status code in the range 200 - 299 indicates a successful operation.

#### Client errors

An HTTP status code in the range 400 - 499 indicates a client error. WebSphere MQ Bridge for HTTP return codes in the range MQHTTP40001 - MQHTTP49999 correspond to client errors.

#### Server errors

An HTTP status code in the range 500 - 599 indicates a server error. WebSphere MQ Bridge for HTTP return codes in the range MQHTTP50001 - MQHTTP59999 correspond to server errors.

If a server error occurs, a complete stack trace is output to the application server error log. The stack trace is also returned to the HTTP client in the HTTP response. Handle the stack trace in the client application or refer it to the application server administrator to resolve the problem.

If the stack trace contains resource adapter errors, refer to the documentation for your resource adapter.

#### Alphabetic list of return codes

##### [HTTP 200: OK](#)

This class of status code indicates that the request was successfully received, understood and accepted.

##### [HTTP 204: No content](#)

Sent following a successful HTTP **GET** or **DELETE** and `x-msg-range: 0` was sent in the request.

##### [MQHTTP0001: No connection factory specified in the Servlet context](#)

Servlet error

##### [MQHTTP0002: Could not get connection manager for queueOrTopic using the JNDI name of jndiNameTried](#)

Servlet error

##### [MQHTTP40001: Reserved](#)

Reserved

##### [MQHTTP40002: URI is not valid for WebSphere MQ transport for HTTP](#)

The URI specified in the HTTP request is not valid.

##### [MQHTTP40003: URI is not valid. @qmgr is only valid on POST](#)

The `@qmgr` URI option has been specified in a URI for an HTTP request that is not a **POST** request.

##### [MQHTTP40004: Invalid Content-Type specified](#)

The `Content-Type` header field specified on a **POST** request is not compatible with the `x-msg-class` header value.

##### [MQHTTP40005: Bad message header value](#)

A supported header field has been specified with a value that is not valid for the specified request.

##### [MQHTTP40006: Header name is not a valid request header](#)

A header that is valid only in an HTTP response message has been specified in an HTTP request message.

##### [MQHTTP40007: Header name is only valid on ...](#)

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

##### [MQHTTP40008: Header name maximum length is ...](#)

The maximum length for the given header field has been exceeded.

##### [MQHTTP40009: Header field header field is not valid for ...](#)

A header field specified in an HTTP request is not supported by the messaging provider to which the WebSphere MQ bridge for HTTP is connected.

##### [MQHTTP40010: Message with Content-Type content type could not be parsed](#)

The content of the HTTP request is not compatible with the `Content-Type` of the request.

##### [MQHTTP40301: You are forbidden from accessing ...](#)

The WebSphere MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

##### [MQHTTP40302: You are forbidden from ...](#)

The WebSphere MQ bridge for HTTP has been unable to connect to the queue manager.

##### [MQHTTP40401: The destination destination name could not be found](#)

The destination specified in the HTTP request URI cannot be found by the WebSphere MQ bridge for HTTP.

##### [MQHTTP40501: Method method namenot allowed](#)

The method specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

##### [MQHTTP41301: The message being posted was too large for the destination](#)

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

##### [MQHTTP41501: The media type character set is unsupported](#)

The character set specified in the `Content-Type` header field is not supported by the WebSphere MQ bridge for HTTP.

##### [MQHTTP41502: Media-type media-type is not supported ...](#)

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified HTTP verb.

##### [MQHTTP41503: Media-type media-type is not supported ...](#)

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified `x-msg-class` header field.

##### [MQHTTP41701: The HTTP header Expect is not supported](#)

The WebSphere MQ bridge for HTTP does not support the `Expect` header field.

##### [MQHTTP50001: There has been an unexpected problem ...](#)



An error has occurred in the WebSphere MQ bridge for HTTP.

**[MQHTTP50201: An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager](#)**

An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager


**[MQHTTP50401: Message retrieval timed out](#)**

No message matching the specified request parameters in an HTTP **GET** or HTTP **DELETE** was returned in the timeout period.

**[MQHTTP50501: HTTP 1.1 and upwards ...](#)**

The HTTP protocol used in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

**Parent topic:** [Reference material for WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts21340\_


## 2.6.5.1. HTTP 200: OK

This class of status code indicates that the request was successfully received, understood and accepted.

### HTTP status code

200 OK

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:09:35

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts200m0\_


## 2.6.5.2. HTTP 204: No content

Sent following a successful HTTP **GET** or **DELETE** and `x-msg-range: 0` was sent in the request.

### HTTP status code

204 No Content

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:09:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts204m0\_

## 2.6.5.3. MQHTTP0001: No connection factory specified in the Servlet context

Servlet error

### Explanation

Servlet error


### HTTP status code

None

### Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:09:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts00001\_

## 2.6.5.4. MQHTTP0002: Could not get connection manager for *queueOrTopic* using the JNDI name of *jndiNameTried*

Servlet error

### Explanation

Servlet error


### HTTP status code

None

### Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:09:34

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts00002\_


### 2.6.5.5. MQHTTP40001: Reserved

Reserved

#### HTTP status code

400 Bad Request

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40001\_

### 2.6.5.6. MQHTTP40002: URI is not valid for WebSphere MQ transport for HTTP

The URI specified in the HTTP request is not valid.

#### Explanation

The URI specified in the HTTP request is not valid.


#### HTTP status code

400 Bad Request

#### Programmer response

Confirm that the format and syntax of the specified URI are correct.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40002\_

### 2.6.5.7. MQHTTP40003: URI is not valid. @qmgr is only valid on POST

The @qmgr URI option has been specified in a URI for an HTTP request that is not a **POST** request.

#### Explanation

The @qmgr URI option has been specified in a URI for an HTTP request that is not a **POST** request.


#### HTTP status code

400 Bad Request

#### Programmer response

If you are attempting to put a message by using the **POST** verb, change the HTTP request to a **POST** request. If you are attempting to get a message by using the **DELETE** or **GET** verbs, remove @qmgr from the URI.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40003\_

### 2.6.5.8. MQHTTP40004: Invalid Content-Type specified

The Content-Type header field specified on a **POST** request is not compatible with the x-msg-class header value.

#### Explanation

The Content-Type header field specified on a **POST** request is not compatible with the x-msg-class header value.


#### HTTP status code

400 Bad Request


#### Programmer response

Change the Content-Type header field to one that is supported. The Content-Type header must be compatible with the specified x-msg-class header field.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40004\_

### 2.6.5.9. MQHTTP40005: Bad message header value

A supported header field has been specified with a value that is not valid for the specified request.

#### Explanation

A supported header field has been specified with a value that is not valid for the specified request.


#### HTTP status code

400 Bad Request


#### Programmer response

Change the value specified for the given header field to a value which is valid. Check the case of the value specified, as some header fields have case-sensitive values.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40005\_

### 2.6.5.10. MQHTTP40006: *Header\_name* is not a valid request header

A header that is valid only in an HTTP response message has been specified in an HTTP request message.

#### Explanation

A header which is valid only in an HTTP response message has been specified in an HTTP request message.


#### HTTP status code

400 Bad Request


#### Programmer response

Remove any headers from the HTTP request which are only valid in an HTTP response; for example, `x-msg-timestamp`.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40006\_

### 2.6.5.11. MQHTTP40007: *Header\_name* is only valid on ...

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

#### Explanation

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

#### HTTP status code

400 Bad Request


#### Programmer response

Remove any headers from the HTTP request which are not valid for the given request verb. For example, `x-msg-encoding` is valid for HTTP **POST** requests, but not valid for HTTP **GET** or HTTP **DELETE** requests.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40007\_

### 2.6.5.12. MQHTTP40008: *Header\_name* maximum length is ...

The maximum length for the given header field has been exceeded.

#### Explanation

The maximum length for the given header field has been exceeded.


#### HTTP status code

400 Bad Request

**Programmer response**

Change the value of the header field to a value which is within the range permitted for the header field.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40008\_

**2.6.5.13. MQHTTP40009: Header field *header\_field* is not valid for ...**

A header field specified in an HTTP request is not supported by the messaging provider to which the WebSphere MQ bridge for HTTP is connected.

**Explanation**

A header field specified in an HTTP request is not supported by the messaging provider to which the WebSphere MQ bridge for HTTP is connected. The error occurs if a messaging provider is used that cannot support all the features of the WebSphere MQ bridge for HTTP.


**HTTP status code**

400 Bad Request

**Programmer response**

Remove the unsupported header from the HTTP request.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:33

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40009\_

**2.6.5.14. MQHTTP40010: Message with Content-Type *content\_type* could not be parsed**

The content of the HTTP request is not compatible with the Content-Type of the request.

**Explanation**

The content of the HTTP request is not compatible with the Content-Type of the request. A common cause is badly formed application/x-www-form-urlencoded or application/xml data.

**HTTP status code**

400 Bad Request

**Programmer response**

Correct the content of the HTTP request so that it is in the correct format for the Content-Type of the request.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:33

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40010\_

**2.6.5.15. MQHTTP40301: You are forbidden from accessing ...**

The WebSphere MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

**Explanation**

The WebSphere MQ bridge for HTTP has been unable to authenticate itself for the specified destination.


**HTTP status code**

403 Forbidden

**Programmer response**

Change the authentication properties of the destination so that the WebSphere MQ Bridge for HTTP is authorized to connect to it. Alternatively, specify a destination to which the WebSphere MQ bridge for HTTP is authorized to connect.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:33

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts40301\_

**2.6.5.16. MQHTTP40302: You are forbidden from ...**

The WebSphere MQ bridge for HTTP has been unable to connect to the queue manager.

**Explanation**

The WebSphere MQ bridge for HTTP has been unable to connect to the queue manager. The WebSphere MQ bridge for HTTP security configuration is incorrect.


**HTTP status code**

403 Forbidden

**Programmer response**

Change the authentication configuration of the queue manager so that the WebSphere MQ Bridge for HTTP is authorized to connect to it. Alternatively, configure the WebSphere MQ bridge for HTTP to connect to a queue manager to which it is authorized to connect.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:33

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts40302\_

## 2.6.5.17. MQHTTP40401: The destination *destination\_name* could not be found

The destination specified in the HTTP request URI cannot be found by the WebSphere MQ bridge for HTTP.

**Explanation**

The destination specified in the HTTP request URI cannot be found by the WebSphere MQ bridge for HTTP.


**HTTP status code**

404 Not found

**Programmer response**

Check that the destination specified in the HTTP request URI exists, or specify an alternative destination.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts40401\_

## 2.6.5.18. MQHTTP40501: Method *method\_name* not allowed

The method specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

**Explanation**

The method specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.


**HTTP status code**

405 Method not allowed

**Programmer response**

Change the method specified in the HTTP request to one which is supported by the WebSphere MQ bridge for HTTP.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts40501\_

## 2.6.5.19. MQHTTP41301: The message being posted was too large for the destination

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

**Explanation**

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.


**HTTP status code**

413 Request entity too large

**Programmer response**

Reduce the size of the message specified in the HTTP request. Alternatively, specify a destination which can support messages of the wanted length.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ts41301\_

### 2.6.5.21. MQHTTP41501: The media type character set is unsupported

The character set specified in the `Content-Type` header field is not supported by the WebSphere MQ bridge for HTTP.

#### Explanation

The character set specified in the `Content-Type` header field is not supported by the WebSphere MQ bridge for HTTP.


#### HTTP status code

415 Unsupported media type

#### Programmer response

Change the character set of the `Content-Type` header field to one that is supported by the WebSphere MQ bridge for HTTP.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts41501\_

### 2.6.5.22. MQHTTP41502: Media-type *media-type* is not supported ...

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified HTTP verb.

#### Explanation

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified HTTP verb.


#### HTTP status code

415 Unsupported media type

#### Programmer response

Change the media-type specified in the HTTP request to one that is supported by the WebSphere MQ Bridge for HTTP for the specified HTTP verb.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts41502\_

### 2.6.5.23. MQHTTP41503: Media-type *media-type* is not supported ...

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified `x-msg-class` header field.

#### Explanation

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified `x-msg-class` header field.


#### HTTP status code

415 Unsupported media type

#### Programmer response

Change the media-type specified in the HTTP request to one that is supported by the WebSphere MQ Bridge for HTTP for the specified `x-msg-class` header field.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts41503\_

### 2.6.5.24. MQHTTP41701: The HTTP header `Expect` is not supported

The WebSphere MQ bridge for HTTP does not support the `Expect` header field.

#### Explanation

The `Expect` header has been specified in an HTTP request. The WebSphere MQ bridge for HTTP does not support the `Expect` header field.


#### HTTP status code

417 Expectation failed

#### Programmer response

Remove the `Expect` header from the HTTP request.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts41701\_

### 2.6.5.25. MQHTTP50001: There has been an unexpected problem ...

An error has occurred in the WebSphere® MQ bridge for HTTP.

#### Explanation

An error has occurred in the WebSphere MQ bridge for HTTP.


#### HTTP status code

500 Internal server error

#### Programmer response

Contact the system administrator of the WebSphere MQ Bridge for HTTP.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts50001\_

### 2.6.5.26. MQHTTP50201: An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager

An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager

#### Explanation

An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager


#### HTTP status code

502 Bad Gateway

#### Programmer response

Contact the system administrator of the WebSphere® MQ Bridge for HTTP.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts50201\_

### 2.6.5.27. MQHTTP50401: Message retrieval timed out

No message matching the specified request parameters in an HTTP **GET** or HTTP **DELETE** was returned in the timeout period.

#### Explanation

No message matching the specified request parameters in an HTTP **GET** or HTTP **DELETE** was returned in the timeout period. The return code indicates that no suitable message was available at any time during the life of the HTTP request.


#### HTTP status code

504 Gateway timeout

#### Programmer response

If a message was expected, check the header fields of the HTTP request such as `x-msg-correlId` and `x-msg-msgid`. Check that the destination specified in the HTTP request URI is correct. Try extending the wait time of the HTTP request using the `x-msg-wait` header field.

**Parent topic:** [HTTP return codes](#)

 This build: January 26, 2011 11:10:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts50401\_

### 2.6.5.28. MQHTTP50501: HTTP 1.1 and upwards ...

The HTTP protocol used in the HTTP request is not supported by the WebSphere® MQ bridge for HTTP.

#### Explanation

The HTTP protocol used in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

#### HTTP status code

505 HTTP version not supported

#### Programmer response

Change the HTTP request to use HTTP protocol V1.1 or higher.

**Parent topic:** [HTTP return codes](#)

This build: January 26, 2011 11:10:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.  
This topic's URL:  
ts50501\_

## 2.6.6. Message types and message mappings for WebSphere Bridge for HTTP

WebSphere® MQ bridge for HTTP supports four message classes, TEXT, BYTES, STREAM and MAP. The message classes are mapped to JMS message types and HTTP Content-Type.

### HTTP POST

The message type that arrives at the destination depends on the value of the x-msg-class header or the Content-Type of the HTTP request. [Table 1](#) shows the HTTP Content-Type type that corresponds to each x-msg-class. Either field can be used to set the message type and message format. If both fields are set, and are set inconsistently, then a Bad Request exception is returned (HTTP 400, MQHTTP20004).

Table 1. Mapping between x-msg-class and HTTP Content-Type

x-msg-class	HTTP Content-Type
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

If the JMS message type is set in the MQRFH2 header, it is mapped according to [Table 2](#).

Table 2. Mapping between x-msg-class and JMS message types.

x-msg-class	JMS message type
BYTES	jms_bytes
TEXT	jms_text
MAP	jms_map
STREAM	jms_stream

The JMS message type is always set for a message class of MAP or STREAM. It is not always set for a message class of BYTES or TEXT. If a MQRFH2 is to be created for the request, the JMS message type is always set. Otherwise, if no MQRFH2 is created, no JMS message type is set. An MQRFH2 is created if user properties are set in the request, using the x-msg-user header.

If the JMS message type is set, then the message format is set to MQFMT\_NONE, see [Table 4](#):

Table 3. Mapping between x-msg-class and WebSphere MQ message format

x-msg-class	Message format with MQRFH2 present in message	Message format with no MQRFH2 present in message
BYTES	MQFMT_NONE	MQFMT_NONE
TEXT	MQFMT_NONE	MQFMT_STRING
MAP	MQFMT_NONE	Not possible
STREAM	MQFMT_NONE	Not possible

### HTTP GET or DELETE

The message type or format retrieved determines the value of the x-msg-class header and the Content-Type of the HTTP response. The x-msg-class header is returned only if requested in a x-msg-headers request.

[Table 4](#) describes the mappings between x-msg-class and Content-Type, and the message type retrieved from the queue or topic.

Table 4. Mapping message types to x-msg-class and Content-Type

Message format	JMS Message type	x-msg-class	Content-Type
Anything except MQFMT_STRING	None	BYTES	application/octet-stream
MQFMT_STRING	None	TEXT	text/plain
MQFMT_NONE	jms_bytes	BYTES	application/octet-stream
MQFMT_NONE	jms_text	TEXT	text/plain
MQFMT_NONE	jms_map	MAP	application/xml
MQFMT_NONE	jms_stream	STREAM	application/xml

### MAP and STREAM message class serialization

MAP and STREAM message classes are serialized back to the client in the HTTP response in the same way as a message is serialized to a queue.

For MAP, the XML name, type, and value triplets are encoded as:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

STREAM is like MAP, but it does not have element names:

```
<stream>
  <elt dt="datatype1">value1</elt>
```




```

    <elt dt="datatype2">value2</elt>
    ...
</stream>

```

**Note:** `datatype` is one of the data types defined for defining user-defined properties and listed in [usr: HTTP x-msg-usr entity-header](#). The attribute `dt="string"` is omitted for string elements because the default data type is `string`.

**Parent topic:** [Reference material for WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts11280\_

## 2.6.7. URI Format

URIs intercepted by WebSphere® MQ bridge for HTTP.

### Syntax

```

>>-http:--/--hostname--+-----+---| Path |-----<<
      |:--port-|

Path

|--/--contextRoot--/------->

>--msg/--+queue/--queueName--+-----+---|-----|
      |                                     |@--qMgrName-| |
      |-topic/--topicName-----|

```

### Note:

1. If a question mark (?) is used it must be substituted with %3f. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. `@qMgrName` is only valid on an HTTP **POST**

### Description

Deploy the WebSphere MQ bridge for HTTP servlet to your JEE application server with a context root of `contextRoot`. Requests to


```
http://hostname:port/context_root/msg/queue/queueName@qMgrName
```

and

```
http://hostname:port/context_root/msg/topic/topicString
```

are intercepted by WebSphere MQ bridge for HTTP.

**Parent topic:** [Reference material for WebSphere MQ bridge for HTTP](#)

 This build: January 26, 2011 11:10:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:  
ts21200\_