# Web Services Routing Protocol (WS-Routing)

**October 23, 2001**

**Authors**

Henrik Frystyk Nielsen
Satish Thatte

## Copyright Notice



## Abstract

Web Services Routing Protocol (WS-Routing) is a SOAP-based, stateless protocol for exchanging one-way SOAP messages from an initial sender to the ultimate receiver, potentially via a set of intermediaries. In addition, WS-Routing provides an optional

reverse message path enabling two-way message exchange patterns like request/response, peer-to-peer conversations, and the return of message acknowledgements and faults. WS-Routing is expressed as a SOAP header entry within a SOAP envelope making it relatively independent of the underlying protocol. This specification defines the use of WS-Routing in combination with TCP, UDP, and HTTP but other underlying protocols are possible.

**Status**

WS-Routing and related specifications are provided as-is and for review and evaluation only. Microsoft hopes to solicit your contributions and suggestions in the near future. Microsoft Corporation makes no warrantees or representations regarding the specifications in any manner whatsoever.

**Table of Contents**


1. Introduction
   1.1 Purpose
2. Notational Conventions
   2.1 Compliance
3. The WS-Routing Model
   3.1 SOAP Message Path Model
   3.2 WS-Routing Message Path Terminology
   3.3 WS-Routing and the SOAP Message Path Model
   3.4 WS-Routing Intermediaries
   3.5 WS-Routing Gateways
   3.6 Fan-in and Fan-out Message Path Models
4. The WS-Routing Mechanisms
   4.1 Endpoints
   4.2 The Forward Message Path
   4.3 The Reverse Message Path
   4.4 Traversing the Forward Message Path
   4.5 Identifying and Correlating Messages
   4.6 Using the Reverse Message Path
5. The WS-Routing Specifications
   5.1 WS-Routing Header Elements
      5.1.1 action
      5.1.2 from
      5.1.3 fwd
      5.1.4 id
      5.1.5 relatesTo
      5.1.6 rev
      5.1.7 to
      5.1.8 via
      5.1.8.1 The vid Attribute
   5.2 WS-Routing Fault Messages
      5.2.1 WS-Routing Sender Faults
      5.2.2 WS-Routing Receiver Faults
   5.3 Use of URIs in WS-Routing

5.4 Timeouts
6. The soap: URI Scheme
6.1 Equivalence Rules
7. WS-Routing Underlying Protocol Bindings
7.1 WS-Routing and DIME
7.2 TCP
7.2.1 TCP Connection Management
7.3 UDP
7.4 HTTP
7.4.1 HTTP Request
7.4.2 HTTP Response
8. Acknowledgement
9. References

---

## 1. Introduction

SOAP Routing Protocol (WS-Routing) is a SOAP-based, stateless protocol for exchanging one-way SOAP messages from an initial sender to the ultimate receiver, potentially via a set of intermediaries. In addition, WS-Routing provides an optional reverse message path enabling two-way message exchange patterns like request/response, peer-to-peer conversations, and the return of message acknowledgements and faults.

SOAP [15] has been designed to be carried within or on top of a variety of other protocols. The boundary between SOAP and the underlying protocol is called a protocol binding. The purpose of a binding is to define the syntactic and semantic interactions between SOAP and the underlying protocol.

SOAP by itself does not define an actual message path along which a SOAP message is to travel. In order to provide the semantics for actually exchanging messages, SOAP can be bound to an application layer protocol such as HTTP or SMTP. These protocols define their own message path models and message exchange patterns that in general differ from the SOAP message model.

In contrast, WS-Routing describes the entire message path within the SOAP message structure using the SOAP extensibility model. This means that a WS-Routing enabled SOAP message does not require a binding to another application layer protocol for describing the message path.

Therefore, in addition to being carried within HTTP, WS-Routing allows a SOAP message to be exchanged directly over transport layer protocols such as TCP and UDP. This specification defines the use of WS-Routing in combination with TCP and UDP as well as HTTP but other protocols are possible.

Note that it is *not* the intent of WS-Routing to provide a complete set of services often considered part of reliable messaging, security, or other services that may be required in a messaging environment. Rather, the design goal is to provide a building block that can be used in combination with other SOAP-based protocols to achieve such goals.

### 1.1 Purpose

**From http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp**

The purpose of WS-Routing is to define the mechanisms needed in order to describe messages being exchanged along the following two message paths:

- A forward message path where messages travel from the initial sender through zero or more intermediaries to the ultimate receiver.
- An optional reverse message path where messages travel in the direction from the ultimate receiver through zero or more intermediaries to the initial sender.

Specifically, this specification defines the following mechanisms and concepts:

- The forward message path (see section 4.2 and 4.4)
- An optional reverse message path (see section 4.3 and 4.6)
- A correlation mechanism between messages (see section 4.5)

Explicit focus has been put on simplicity and applicability of the protocol in the same way SOAP itself is widely applicable. This means that the mechanisms provided by this specification have been deliberately limited to serve the purposes stated above.

## 2. Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

The WS-Routing header entry is identified by the XML Namespace URI "http://schemas.xmlsoap.org/rp". The elements defined by WS-Routing making up the SOAP header entry are defined by XML schema [10][11].

### 2.1 Compliance

An implementation is not WS-Routing compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined in this specification.

A WS-Routing receiver that accepts incoming WS-Routing messages MUST be WS-Routing compliant. A WS-Routing sender MUST NOT use the "http://schemas.xmlsoap.org/rp" XML Namespace identifier [8] unless it is WS-Routing compliant.

## 3. The WS-Routing Model

This chapter describes the WS-Routing model, which is based on the SOAP message path model described in section 3.1. The message path terminology used by WS-Routing is described in section 3.2, and the WS-Routing model is described in the remainder of this chapter.

### 3.1 SOAP Message Path Model

SOAP is based on a stateless, one-way message model defined in terms of SOAP senders and SOAP receivers that can send and receive SOAP messages respectively. In addition SOAP has the notion of intermediaries that can act as both senders and receivers.

Intermediaries are central to SOAP in that the SOAP message model provides a distributed processing mechanism in which the SOAP "actor" attribute can be used to indicate which part of a message is intended for a given SOAP receiver. The purpose of this model is to provide a flexible mechanism for composing and using distributed value-added services such as annotation, collaboration, subscription, privacy enforcement, and caching. By allowing these services to be composed dynamically across network nodes, the model makes it possible to introduce new services for the proliferation of devices and end users' data (see [13], [14]).

Despite the implied SOAP message model, SOAP does *not* define a mechanism for indicating the SOAP senders and receivers along the SOAP message path or the order in which the senders and receivers are composed. In short, SOAP does not define a "message path". For example, take four SOAP processors **A**, **B**, **C**, and **D**. A SOAP message generated by **A** can indicate which part of a message is for **B**, **C,** and **D**. However, it cannot indicate that the intermediaries are to be organized into a message path illustrated in Figure 1.

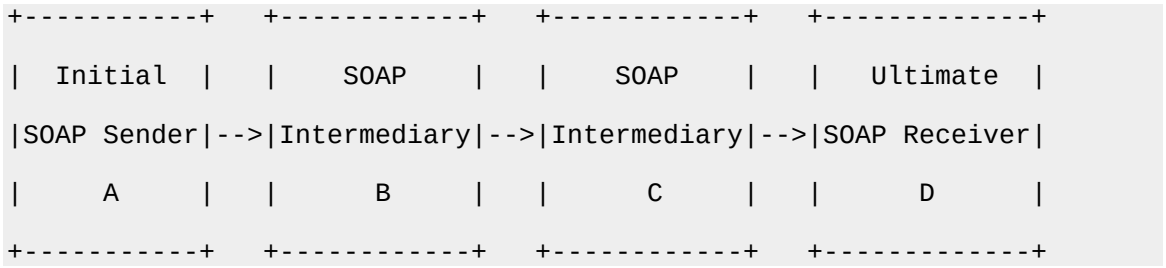


**Figure 1** SOAP defines which parts of a message are for **B**, **C**, and **D** but not that the message is to travel from **A** via **B** via **C** to **D**.

In order to provide the semantics for actually exchanging messages, SOAP can be bound to other application layer protocols such as HTTP and SMTP. However, these protocols define their own message path models and message exchange patterns, which differ from the general SOAP model. As a result, it is not possible to use these protocol bindings alone to describe an exchange of a SOAP message from **A** to **D**.

WS-Routing on the other hand, defines a message path model that is fully compatible with the SOAP message-processing model. In other words, WS-Routing makes it possible to describe the complete exchange of a SOAP message from **A** to **D** in Figure 1 and to describe which parts of a SOAP message is intended for what SOAP receiver in the message path.

### 3.2 WS-Routing Message Path Terminology

The WS-Routing message path terminology is based on the SOAP message model terminology. However, as not all SOAP messages are WS-Routing messages, an explicit distinction is made between SOAP messages in general and WS-Routing-enabled SOAP messages in particular. WS-Routing defines the following set of specific terms, which are used throughout this document:

WS-Routing header
: A SOAP/1.1 header entry identified by the "http://schemas.xmlsoap.org/rp" XML namespace URI and which complies with this specification (see section 2.1).

WS-Routing message
: A SOAP/1.1 message that contains a WS-Routing header composed in compliance with this specification (see section 2.1).

WS-Routing fault message

A SOAP/1.1 fault message that contains a WS-Routing header composed in compliance with this specification (see section 2.1 and section 5.2).

WS-Routing sender
: An application that generates a WS-Routing message and binds to a specific underlying protocol for the purpose of transferring the message.

WS-Routing receiver
: An application that accepts an incoming WS-Routing message transferred using some underlying protocol for the purpose of processing the message.

WS-Routing message path
: The set of WS-Routing senders and WS-Routing receivers that process a single WS-Routing message when it is exchanged. WS-Routing defines a forward message path and a reverse message path.

forward WS-Routing message path
: The message path used by a WS-Routing message traveling from the initial WS-Routing sender, through zero or more WS-Routing intermediaries, and to the ultimate WS-Routing receiver.

reverse WS-Routing message path
: An optional message path that is built dynamically when a WS-Routing message is sent along the forward message path. The reverse message path can be used for sending one or more WS-Routing messages to the initial WS-Routing sender of the message for which the reverse path was built.

initial WS-Routing sender
: The WS-Routing sender that originates a WS-Routing message as the starting point of a WS-Routing message path.

WS-Routing intermediary
: An application that can act as both a WS-Routing sender and a WS-Routing receiver with the purpose of processing and forwarding a WS-Routing message along a WS-Routing message path. A WS-Routing intermediary can take on one of two roles: proxy and tunnel (see section 3.4).

WS-Routing proxy
: A WS-Routing proxy is a WS-Routing intermediary that has taken on the role of serving an incoming message in which the proxy takes full part in the WS-Routing communication in accordance with section 4.4 and 4.6.

WS-Routing tunnel
: An intermediary acting as a blind communication relay between two underlying communication channels. Once active, a tunnel is not considered a part of the WS-Routing communication. The tunnel ceases to exist when both the relayed communication channels are closed. A WS-Routing sender sending a message to an intermediary MUST explicitly request the tunnel role to be used. A WS-Routing intermediary MUST NOT use the tunnel role if not explicitly requested. WS-Routing does not define a mechanism for requesting a tunnel as it is expected to be defined in a separate specification.

ultimate WS-Routing receiver
: The WS-Routing receiver that the initial sender specifies as the final destination of the WS-Routing message within a WS-Routing message path. In case of a WS-Routing fault along the message path, a WS-Routing message may not reach the ultimate receiver.

WS-Routing gateway
: An ultimate WS-Routing receiver that performs a semantic mapping of a WS-Routing header, potentially into some other protocol unknown to SOAP and

WS-Routing. An example would be a WS-Routing-->Telnet protocol translation of the message (see section 3.5).

### 3.3 WS-Routing and the SOAP Message Path Model

This specification defines a single SOAP header entry, which describes a forward and optionally a reverse message path for a SOAP message for the purpose of exchanging messages along these message paths.

The WS-Routing design is based on the following model of communication: a message may be initiated for any reason defined by some application using WS-Routing. The initial WS-Routing sender indicates the ultimate WS-Routing receiver of the message and zero or more WS-Routing intermediaries (see section 3.4) in the forward message path in the WS-Routing header. The message is transferred to the first WS-Routing receiver in the forward message path over some underlying protocol. If that WS-Routing receiver is an intermediary, it forwards the message to the next WS-Routing receiver in the forward message path until it reaches the ultimate WS-Routing receiver or a fault occurs.

Given certain restrictions, WS-Routing intermediaries may dynamically insert additional intermediaries in the forward message path (see section 3.4). That is, there is no requirement that the complete forward message be known at the time it leaves the initial WS-Routing sender.

In addition to the forward message path, WS-Routing defines an optional reverse message path that enables two-way message exchange patterns like request/response and peer-to-peer conversations as well as returning message acknowledgements and faults. The reverse path is built dynamically as a message flows in the forward direction (see section 4.6 and 5.2).

The purpose of the reverse path is to indicate a possible path for one or more messages to be sent to the initial WS-Routing sender of the message for which the reverse path was built. The WS-Routing message path model is still that of a one-way message but the reverse path allows a recipient of a message to send another message back to the initial WS-Routing sender without having to create the concept of a channel or virtual connection between the two parties.

There is no requirement that the reverse path actually be used nor does the reverse path enforce any particular message exchange pattern. If the reverse path is being used to exchange a message, the reverse path becomes the forward path and the exact same rules apply for the exchange as for any other message. This means that the generator of the message always is the initial WS-Routing sender and the final destination always is the ultimate WS-Routing receiver regardless of whether the forward message path is a reverse message path of some other message.

Intermediaries may short-circuit the forward message path and, if appropriate, generate a message and send it along the reverse path. This can for instance be a cached version of a previous reply message. It is not within the scope of WS-Routing to specify when or how such a reverse path message is appropriate. The only reverse path messages defined by WS-Routing are WS-Routing fault messages (see section 5.2).

Any given underlying protocol that WS-Routing is bound to may provide a bidirectional communication channel (TCP for example) or a unidirectional communication channel (UDP for example). Depending on the WS-Routing protocol binding, reverse path messages may or may not be transferred on the same communication channel.

Regardless of whether the communication channel is unidirectional or bidirectional, it is possible that a WS-Routing sender may wish to indicate a different endpoint for the optional reverse path. WS-Routing defines a reverse path that can be either explicitly specified independently of the underlying protocol, or implicitly specified as a function of the underlying protocol (see section 5.1.8). That is, *if* the protocol binding provides a bidirectional channel then the implicit reverse message path is defined as being that channel.

In some cases, it may be desirable to switch between protocol bindings along a given message path (see section 3.5). For example, for optimization purposes, it might be desirable to use TCP on some parts and UDP on other parts of a message path. WS-Routing does not itself define any security or reliability services, which means that there are no means within WS-Routing to compensate for different levels of services provided by the underlying protocol. It is expected that other SOAP-based protocols will provide policies and mechanisms for end-to-end quality of service guarantees.

**From http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp**

### 3.4 WS-Routing Intermediaries

WS-Routing intermediaries can process and forward WS-Routing messages on behalf of other WS-Routing senders. WS-Routing defines the terms "proxy" and "tunnel" (see section 3.2) as the two possible roles that a WS-Routing intermediary can take on in order to serve an incoming message.

The purpose of the WS-Routing intermediaries is to provide a mechanism for allowing WS-Routing applications to

- access value-added services that the WS-Routing sender may have subscribed to or is otherwise using. Examples of such services are annotation services, collaboration services, subscription management services, privacy enforcement services, caching services, etc.
- facilitate traversal of application-level intermediaries such as application relay firewalls that have been put in place for administrative or policy purposes.

Given the restrictions below, it is possible to insert additional intermediaries in the forward message path without renegotiating the message path with the previous WS-Routing sender. A WS-Routing intermediary intending to insert additional intermediaries in the forward message path MAY do so if it fulfills the following criteria:

- it SHOULD NOT insert intermediaries unless it has administrative reasons for doing so or is using a value-added service provided by those intermediaries;
- it MUST NOT insert intermediaries immediately *before* an empty "via" element in the forward message path as empty "via" elements indicate an implicit reverse message path defined by the underlying protocol (see section 5.1.8);
- it SHOULD be careful not to introduce infinite routing loops when adding intermediaries to the message path.

A WS-Routing intermediary MUST NOT rewrite the "to" element of a WS-Routing message (see section 5.1.7).

A WS-Routing intermediary MAY use a different underlying protocol binding for the sending and receiving side. An example of a change in protocol bindings is WS-

Routing/TCP-->WS-Routing/UDP as illustrated in Figure 2 (see section section 7 for the WS-Routing bindings to TCP and UDP):

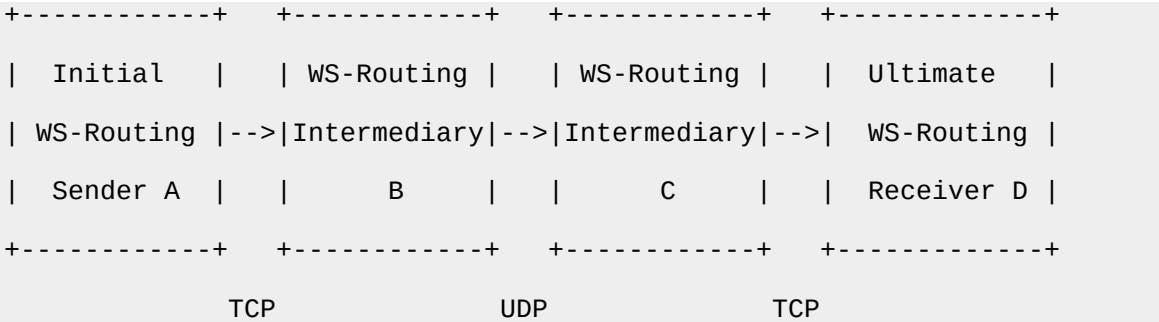


**Figure 2** Examples of possible changes in underlying protocols

Note again, that WS-Routing does not define any security or reliability services, which means that there are no means within WS-Routing to compensate for different levels of services provided by the underlying protocol. It is expected that other SOAP-based protocols will provide policies and mechanisms for end-to-end quality of service guarantees.

### 3.5 WS-Routing Gateways

A WS-Routing receiver can act as a gateway *into* a foreign protocol environment. A WS-Routing sender sending a message to a WS-Routing receiver acting as a gateway may not know that it is communicating with a WS-Routing gateway. Similarly, a WS-Routing sender can act as a gateway *from* a foreign protocol environment into WS-Routing and likewise, a WS-Routing receiver receiving a message from a WS-Routing sender acting as a gateway may not know that it is communicating with a WS-Routing gateway.

WS-Routing gateways SHOULD take into account proper migration of fault codes across the protocol boundaries. If the foreign protocol environment has no equivalent concept of faults, the gateway SHOULD select and use a best approximation of WS-Routing fault codes.

A WS-Routing intermediary that also acts as a WS-Routing gateway MUST follow the rules of a WS-Routing intermediary and MUST perform the gateway functionality as part of both the WS-Routing sender and the WS-Routing receiver in order to forward the message along the WS-Routing message path (see also section 3.4).

For the purpose of this specification, a WS-Routing receiver that rewrites the "to" element of a WS-Routing message is considered a WS-Routing gateway (see section 5.1.7). A WS-Routing intermediary that uses a different underlying protocol binding on the sending and receiving side is *not* considered a gateway although it may still have to map fault codes across the protocol binding boundaries.

### 3.6 Fan-in and Fan-out Message Path Models

WS-Routing does not define any fan-in or fan-out message path models, which may be required to encapsulate the semantics of more complex multi-party interactions. It is not a design goal of this specification to directly support these models as it is anticipated that other message routing specifications will define support for such patterns.

## 4. The WS-Routing Mechanisms

This section introduces the mechanisms used in WS-Routing for describing the forward message path as well as the reverse message path and how to control these paths.

### 4.1 Endpoints

WS-Routing defines the following elements to indicate the senders and receivers in a message path:

to
: Identifies the ultimate WS-Routing receiver by means of a URI (see section 5.1.7).

via
: Identifies by means of a URI an intermediary that the message must pass along the message path (see section 5.1.8 for a discussion of the possible values of the "via" element).

from
: Identifies the party originating the message by means of a URI (see section 5.1.2).

Note, that in WS-Routing both the endpoint for which the message is intended as well as any intermediary that the message is to go through is identified by URIs. The "to" element identifies the resource for which the message is intended similar to the HTTP Request-URI (see [6] section 5.1.2) and the "via" elements identify the intermediaries along the message path.

### 4.2 The Forward Message Path

WS-Routing defines the following element for describing the forward message path:

fwd
: Describes the forward message path as an ordered list of "via" elements indicating the intermediaries that the message must go through (see section 5.1.6).

The forward path can be constructed as the message moves along the message path or at the initial SOAP sender (see section 4.4).

### 4.3 The Reverse Message Path

WS-Routing defines the following element for describing an optional reverse message path that can be used to transfer messages to the initial WS-Routing sender if requested:

rev

Describes the reverse message path as an ordered list of "via" elements indicating the reverse path that reverse path messages must go through (see section 5.1.6).

If a "rev" element is present in a message then the reverse path is constructed as the message moves along the message path as described in section 4.4 and section 4.6.

### 4.4 Traversing the Forward Message Path

This section describes the algorithm for traversing a forward message path. The following rules ensure that a message can travel from the initial WS-Routing sender to the ultimate WS-Routing receiver:

1. The initial WS-Routing sender MUST generate a WS-Routing "path" header that indicates the route to be taken by the message. The path MAY indicate a route via one or more intermediaries using the "via" elements as sub-elements of the "fwd" element. The initial sender MAY indicate the ultimate destination by using a "to" element. In the absence of a "to" element the ultimate destination is indicated by the last "via" in the "fwd" element. The second option occurs most commonly when an ultimate destination reverses roles, becomes an initial sender, and uses the reverse path in a received message as a forward path to send a response to the original sender.

   In addition the initial WS-Routing sender MAY insert a reverse path for indicating where the initial sender can receive reverse path messages. The initial sender sets the ultimate destination in the reverse path using a "via" element as a sub-element of the "rev" element.

2. A WS-Routing receiver receiving a WS-Routing message MUST inspect the WS-Routing header and perform the following operations:

   - If no "fwd" element is present or if the "fwd" element does not contain any "via" elements then inspect the "to" element and verify that the value identifies THIS WS-Routing receiver. If this is the case then THIS WS-Routing receiver is the ultimate destination. If there is no "to" element or if the value of the "to" element does not identify THIS WS-Routing receiver then generate a fault (see section 5.2).

   - If the "fwd" element is present and contains one or more "via" element(s) then remove the top "via" element listed in the "fwd" element and verify that the value of that "via" element is either empty or identifies THIS WS-Routing receiver or failing that generate an appropriate WS-Routing fault (see section 5.2). If, after removing the top "via" element there are no remaining "via" element(s) listed in the "fwd" element, and there is no "to" element, then THIS WS-Routing receiver is the ultimate destination.

3. A WS-Routing intermediary MUST follow these additional rules:

- A WS-Routing intermediary MAY add to the remaining message path given the restrictions listed in section 3.4.
- If a "rev" element is present then add a "via" element as the first "via" element listed in the "rev" element with a value indicating the reverse path endpoint. If a reverse path endpoint cannot be provided then generate a 751 "Reverse Path Unavailable" WS-Routing fault (see section 5.2).
- If one or more "via" element(s) remain in the "fwd" element then forward the WS-Routing message to the endpoint identified by the new top "via" element listed in the "fwd" element.
- If there are no remaining "via" element(s) listed in the "fwd" element but there is a "to" element then forward the WS-Routing message to the endpoint identified by the "to" element.
- In the last two cases if the forwarding does not succeed then generate the appropriate WS-Routing fault (see section 5.2).

Note that the list of "via" elements in the forward message path is ordered and multiple WS-Routing processors along it may or may not reside physically at the same host. The ordering must always be respected for processing regardless. For instance, it is possible that the last WS-Routing intermediary and the ultimate WS-Routing receiver both reside at the same host. The algorithm above ensures that the intermediary role is executed before the ultimate receiver role.

The example below, Example 1, illustrates a very simple message path with no intermediaries and no reverse path.

**Example 1** Minimalist message with no reverse path and no intermediaries in the message path

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.notification.org/update</m:action>
         <m:to>soap://notification.com/some/endpoint</m:to>
```

```
         <m:id>uuid:09233523-345b-4351-b623-5dsf35sgs5d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

Below is a set of examples illustrating a message traversing the forward message path depicted in Figure 1:

**Example 2** Message leaving initial WS-Routing Sender A in forward direction towards B with a reverse path

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://D.com/some/endpoint</m:to>
         <m:fwd>
            <m:via>soap://B.com</m:via>
            <m:via>soap://C.com</m:via>
         </m:fwd>
         <m:rev>
            <m:via/>
         </m:rev>
         <m:from>mailto:henrikn@microsoft.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
```

```
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

**Example 3** Message leaving intermediary B in forward direction towards **C** with a reverse path

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://D.com/some/endpoint</m:to>
         <m:fwd>
            <m:via>soap://C.com</m:via>
         </m:fwd>
         <m:rev>
            <m:via/>
            <m:via m:vid="cid:122326@B.com"/>
         </m:rev>
         <m:from>mailto:henrikn@microsoft.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

An empty "via" element indicates that the reverse path is provided by the underlying protocol, i.e. the underlying protocol provides a bidirectional communication channel (see section 5.1.8).

Note that intermediary B in Example 3 has added an optional "vid" attribute to the empty "via" element inserted by the original sender **A** (see section 5.1.8 and 5.1.8.1). The "vid" attribute enables an intermediary such as **B** to identify the bidirectional channel back to **A** without maintaining per message state and without requiring a one-to-one correspondence between the bidirectional channel **B** uses to reach **C** and the bidirectional channel **A** used to reach **B**. This allows the management of channels between **B** and **C** to happen completely independent of the management between **A** and **B** (see section 7.2.1).

In the next example, **C** chooses to preserve the empty "via" element inserted by **B** for the reverse path from **C** to **B** because it is able to correlate the UDP endpoint it inserts for the reverse path to itself from **D** with the connection to be used to continue the reverse path to **B**.

**Example 4** Message leaving intermediary C in forward direction towards D with a reverse path

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/chat</m:action>
         <m:to>soap://D.com/some/endpoint</m:to>
         <m:fwd>
         </m:fwd>
         <m:rev>
            <m:via>soap://C.com/rev/endpoint1;up=udp</m:via>
            <m:via/>
            <m:via m:vid="cid:122326@B.com"/>
         </m:rev>
```

```
         <m:from>mailto:henrikn@microsoft.com</m:from>
         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

### 4.5 Identifying and Correlating Messages

Being able to identify a message and to correlate that message with other messages is essential to WS-Routing. The correlation can for example be between multiple messages flowing in the same direction on either the forward or the reverse message path or it can be between messages on different message paths. An example of a correlation is between a WS-Routing fault message and the faulty message. WS-Routing defines the following two elements to enable this correlation:

**id**
: The value of the "id" element is a unique identifier in the form of a URI that identifies THIS message (see section 5.1.4).

**relatesTo**
: The value of the "relatesTo" element is the value of the "id" element of the message that this message is related to. A message can have any number of related messages (see section 5.1.5).

Section 5.1.4 and 5.1.5 describe in more detail the problem of defining uniqueness.

### 4.6 Using the Reverse Message Path

The optional reverse message path can be used for sending one or more WS-Routing messages to the initial WS-Routing sender of the message for which the reverse path was built. The reverse message path can be used for two-way message exchange patterns like request/response, peer-to-peer conversation as well as returning message acknowledgements and faults.

If a "rev" element is present in a message, the reverse message path is built dynamically when a WS-Routing message travels along the forward message path (see section 4.4). Only an initial WS-Routing sender MAY insert a "rev" element into

a message. If the WS-Routing receiver is an intermediary and a reverse path cannot be established to the next WS-Routing receiver in the message path then it MUST generate an 751 "Reverse Path Unavailable" WS-Routing fault (see section 5.2). A WS-Routing receiver MUST NOT remove the "rev" element from a message.

If the reverse message path of a WS-Routing message is to be used for sending another message, the ordered list of "via" elements in the "rev" element is used as the list of "via" elements in the "fwd" element as described in section 4.4. A WS-Routing message being exchanged using a reverse message path is in all respects like any other WS-Routing message and follows the exact same rules as described in section 4.4.

Note that there is no requirement that the reverse path actually be used nor is there a requirement that reverse path messages be sent in the same order as messages in the forward path or even that there is an equal number of messages flowing in either direction (see section 4.5 for how to correlate WS-Routing messages).

A message being exchanged in the reverse message path can itself define a reverse path allowing for message dialogs. In order to avoid "fault loops", a WS-Routing fault message MUST NOT be sent in response to another WS-Routing fault message (see section 5.2).

Below is a set of examples illustrating a message traversing the reverse message path built in Examples 2-4. In Example 5, **D** sends the message to the endpoint defined by **C** for the reverse path. As this endpoint uses UDP, which is not a bidirectional channel, **D** cannot use an empty "via" element in its reverse path.

> **Example 5** Reverse path message leaving ultimate WS-Routing receiver D headed for explicit reverse path endpoint at C

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/update</m:action>
         <m:fwd>
```

```
            <m:via>soap://C.com/rev/endpoint1;up=udp</m:via>
            <m:via/>
            <m:via m:vid="cid:122326@B.com"/>
         </m:fwd>
         <m:rev>
            <m:via>soap://D.com/some/endpoint</m:via>
         </m:rev>
         <m:from>mailto:satisht@microsoft.com</m:from>
         <m:id>uuid:9fshs8fj-sffg-r5ts-adfg-9kd84jd9mjdld43</m:id>
         <m:relatesTo>uuid:84b9f5d0-33fb-4a81-b02b-
5b760641c1d6</m:relatesTo>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

**Example 6** Reverse path message leaving intermediary C headed for implicit reverse path endpoint at B

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/update</m:action>
         <m:fwd>
            <m:via/>
            <m:via m:vid="cid:122326@B.com"/>
         </m:fwd>
         <m:rev>
```

```
            <m:via>soap://C.com/rev/endpoint2;up=udp</m:via>
            <m:via>soap://D.com/some/endpoint</m:via>
         </m:rev>
         <m:from>mailto:satisht@microsoft.com</m:from>
         <m:id>uuid:9fshs8fj-sffg-r5ts-adfg-9kd84jd9mjdld43</m:id>
         <m:relatesTo>uuid:84b9f5d0-33fb-4a81-b02b-
5b760641c1d6</m:relatesTo>
      </m:path>
   </S:Header>
   <S:Body>
      ...
   </S:Body>
</S:Envelope>
```

When the messages reach **B**, it can recover the value of the "vid" attribute that it inserted in the forward message path and find the correct transport channel for sending the message to **A**. As the value of the "vid" attribute is meaningful to **B** only it MUST take out the value before forwarding the message to **A**.

> **Example 7** Reverse path message leaving intermediary B headed for implicit reverse path endpoint at A

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://www.im.org/update</m:action>
         <m:fwd>
            <m:via/>
         </m:fwd>
         <m:rev>
            <m:via/>
```

```
            <m:via>soap://C.com/rev/endpoint2;up=udp</m:via>

            <m:via>soap://D.com/some/endpoint</m:via>

         </m:rev>

         <m:from>mailto:satisht@microsoft.com</m:from>

         <m:id>uuid:9fshs8fj-sffg-r5ts-adfg-9kd84jd9mjdld43</m:id>

         <m:relatesTo>uuid:84b9f5d0-33fb-4a81-b02b-
5b760641c1d6</m:relatesTo>

      </m:path>

   </S:Header>

   <S:Body>

      ...

   </S:Body>

</S:Envelope>
```

When the message arrives at **A**, the reverse path contains a possible forward path for sending another message to **D**.

## 5. The WS-Routing Specifications

### 5.1 WS-Routing Header Elements

All element content values in WS-Routing are defined as URIs without exception. This section defines the WS-Routing elements in which URIs are used and section 5.3 defines the rules for handling URIs as values in WS-Routing.

WS-Routing header elements not defined by this specification SHOULD be ignored by a WS-Routing receiver and MUST be forwarded by WS-Routing intermediaries.

#### 5.1.1 action

The "action" element is used to indicate the intent of the WS-Routing message in a manner similar to the SOAPAction HTTP header field defined for SOAP (see [15], section 6.1.1). The value is a URI identifying the intent. Similar to the SOAPAction header field, WS-Routing places no restrictions on the format or specificity of the URI or requires that it can be dereferenced. There is no mechanism for computing the value based on the message and there is no default value.

An explicit "action" element MUST be present in all WS-Routing messages. It MUST be generated by the initial WS-Routing sender and MUST NOT be modified along the message path. WS-Routing defines a specific value that MUST be used to identify WS-Routing fault messages as defined by this specification:

WS-Routing fault messages
: The value "http://schemas.xmlsoap.org/soap/fault" MUST be used when the message is a SOAP fault message (see section 3.2 and 5.2) regardless of whether it is carrying a WS-Routing fault or some other SOAP fault.

#### 5.1.2 from

The "from" element can be used to identify the entity or human who is responsible for the message. It MAY be generated by an initial WS-Routing sender. It MUST NOT be generated by any other WS-Routing sender.

The value of a "from" element, if given, is a URI and typically it will be in the form of a "mailto:" URI. The semantics of the "from" element is similar to that of the HTTP "From" header field (see [6], section 14.22).

A WS-Routing receiver MAY use the "from" element for logging purposes and as a means for identifying the person responsible for the message. It SHOULD NOT be used as an insecure form of access protection nor as a trusted identity mechanism.

Sending a "from" element can have privacy implications that may be in conflict with the user's privacy interests. It is expected that initiatives like P3P can provide a mechanism for describing a user's privacy policy.

#### 5.1.3 fwd

The "fwd" element contains an ordered list of intermediaries that the message is to pass on the forward message path. Each intermediary is indicated using the "via" element and is listed in the top-down order in which they are to be contacted. The mechanism used for traversing the message path is described in section 4.4.

#### 5.1.4 id

The "id" element MUST be used to uniquely identify a message. It MUST be generated by the initial WS-Routing sender and MUST NOT be modified along the message path. The value of the element is a unique identifier in the form of a URI that refers to THIS message and this message only.

An "id" value MUST NOT be reused in any other WS-Routing message regardless of whether the message is being exchanged via the forward or the reverse message path.

Examples of "id" element values are:

```
uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6

mid:C10F7F15B07@bar.net
```

The uniqueness of the message identifier is guaranteed by the initial WS-Routing sender. It is STRONGLY RECOMMENDED that the identifier is as unique over space and time as possible and that identity values are either Universally Unique Identifiers (UUIDs), as illustrated in the example above or generated from message content using cryptographic hash algorithms such as MD5.

### 5.1.5 relatesTo

The "relatesTo" element MAY be used in any message to indicate that the message is related to another message in some manner. The element can be used to indicate relationships between two messages regardless of whether they use the same message path or not. Multiple messages can be related to the same message allowing for supporting a variety of message exchange patterns.

It is not the intent of the "relatesTo" element to indicate *how* two messages are related—the exact relationship is expected to be defined elsewhere in the message. If used, the element MUST be generated by the initial WS-Routing sender and the element value MUST be the exact value of the corresponding "id" element. Below is an example of a "relatesTo" element:

```
<relatesTo>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</relatesTo>
```

WS-Routing fault messages MUST use the "relatesTo" element to indicate the relationship between the WS-Routing fault and the faulty message (see section 5.2).

#### 5.1.6 rev

The "rev" element describes the reverse path that reverse path messages must follow. Each party in the reverse message path including the initial WS-Routing sender is listed in a top-down order using the "via" element. The mechanism used for traversing a message path is described in section 4.4.

A "rev" element MAY be inserted into a message by the initial WS-Routing sender. It MUST NOT be inserted by any other party in the message path.

#### 5.1.7 to

The "to" element is used to indicate the ultimate destination of the message in a way similar to the HTTP Request-URI. However, it is important to note that there is no request/response message exchange pattern implied by its use in WS-Routing.

Unless a message follows the reverse message path of another message, the initial WS-Routing sender MAY insert a "to" element into a message. A reverse path message MUST NOT contain a "to" element as the ultimate destination is already indicated as part of the reverse message path (see section 5.1.6).

The value of the "to" element MUST be determined by the initial WS-Routing sender. Intermediaries MUST NOT modify the value.

#### 5.1.8 via

The "via" element is used to indicate that a message is to go through a WS-Routing receiver identified by the value of the "via" element. It can be inserted by any WS-Routing sender in the message path following the rules described in section 3.4.

The value of a "via" element can be either empty or an absolute URI (see section 5.3). A non-empty value indicates the explicit endpoint of the WS-Routing receiver that is to receive the message at a given point in the forward message path.

An empty value indicates an implicit endpoint provided by the underlying protocol binding, typically as a result of a previous message having built a reverse message path using that underlying communication channel. The mechanism used for traversing the forward and reverse message path is described in section 4.4 and 4.6 respectively.

##### 5.1.8.1 The vid Attribute

A WS-Routing intermediary MAY insert an "vid" attribute on the topmost "via" element in the reverse path of the message that it receives if that "via" element is empty. As stated above, an empty "via" element indicates that the message path is provided by the underlying protocol, i.e. the underlying protocol provides a bidirectional communication channel.

The purpose of the "vid" attribute is to enable an intermediary to identify the bidirectional channel to the previous WS-Routing sender without maintaining per message state and without requiring a one-to-one correspondence between the bidirectional channel to the previous WS-Routing sender and the next WS-Routing receiver in the forward message path. This allows a WS-Routing intermediary to separate the management of bidirectional channels to the previous WS-Routing sender and the next WS-Routing receiver in the message path (see section 7.2.1).

The value of the "vid" attribute is a unique URI that can be used by the generator of the value to correlate between the bidirectional communication channel indicated by the empty "via" element and messages using the reverse path.

Examples of "vid" element values are:

```
cid:C10F7B07@bar.net

uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6
```

The uniqueness of the value of the "vid" attribute is guaranteed by the WS-Routing intermediary generating it. It is STRONGLY RECOMMENDED that the identifier is as unique over space and time as possible.

### 5.2 WS-Routing Fault Messages

WS-Routing defines a variety of fault messages indicating problems that can occur along the message path. Any WS-Routing receiver along the message path MAY generate a WS-Routing fault.

If there is a reverse message path present in the faulty WS-Routing message then the WS-Routing fault SHOULD be returned to the initial WS-Routing sender. If the message path is not bidirectional then the fault message SHOULD be discarded.

WS-Routing fault messages MUST use the "relatesTo" element to indicate the relationship between the WS-Routing fault and the faulty message (see section 5.1.5).

A WS-Routing fault MUST NOT be exchanged in response to another WS-Routing fault message as this can result in infinite "fault loops". WS-Routing faults generated in response to other WS-Routing faults MUST be silently discarded.

WS-Routing faults are carried within the WS-Routing header together with a SOAP fault message in the WS-Routing message body. The WS-Routing header contains the WS-Routing fault information. WS-Routing defines two classes of fault codes: "Sender" faults and "Receiver" faults that match the SOAP fault codes "Client" and "Server".

The term "Sender" and "Receiver" is always relative to the message path. For example, in Example 3, the Sender is **B** and the Receiver is **C**, and in Example 6, the Sender is **C** and the Receiver is **B**.

A WS-Routing fault element contains two sub-elements:

code
: The value of the "code" element is a number defined by this specification indicating the WS-Routing specific fault code.

reason
: The value of the "reason" element is a phrase explaining the WS-Routing fault code intended for human consumption. This specification suggests English phrases but there is no requirement that these phrases be used.

In addition, the fault element can contain information specific to individual WS-Routing fault codes as defined in section 5.2.1 and 5.2.2.

**Example 9** An example of a WS-Routing fault message generated by ultimate WS-Routing recipient D with a WS-Routing fault element

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://schemas.xmlsoap.org/soap/fault</m:action>
         <m:fwd>
            <m:via>soap://C.com/rev/endpoint1/</m:via>
            <m:via/>
            <m:via/>
         </m:fwd>
         <m:rev>
         </m:rev>
         <m:from>mailto:satisht@microsoft.com</m:from>
         <m:id>mid:C10F7F33B880B248BC8470115B07@bar.net</m:id>
         <m:relatesTo>uuid:67823759-45f3-45ds-56g6-
45fw45wg66sf</m:relatesTo>
         <m:fault>
            <m:code>812</m:code>
            <m:reason>Service Too Busy</m:reason>
            <m:retryAfter>300</m:retryAfter>
         </m:fault>
      </m:path>
```

```
   </S:Header>
   <S:Body>
      <S:Fault>
         <S:faultcode>S:Server</S:faultcode>
         <S:faultstring>Server Error</S:faultstring>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

An intermediary can generate a fault message indicating that the message cannot be sent further along the message path.

> **Example 10** This is a fault message generated by intermediary C sent via the reverse message path similar to Example 4 but where a bidirectional message path is available. The fault message is returned to the initial WS-Routing sender, **A**, instead of being forwarded to the ultimate WS-Routing receiver, **D**.

```
<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
         <m:action>http://schemas.xmlsoap.org/soap/fault</m:action>
         <m:fwd>
            <m:via/>
            <m:via/>
         </m:fwd>
         <m:rev>
         </m:rev>
         <m:id>uuid:847sjsk5-gdfa-rdg2-45as-87ahsja8mjdld43</m:id>
         <m:relatesTo>uuid:1254sd65-44f3-45d4-5345-
4t3w4984621e</m:relatesTo>
         <m:fault>
```

```
            <m:code>710</m:code>
            <m:reason>Endpoint Not Found</m:reason>
            <m:endpoint>soap://D.com/some/endpoint</m:endpoint>
         </m:fault>
      </m:path>
   </S:Header>
   <S:Body>
      <S:Fault>
         <S:faultcode>S:Client</S:faultcode>
         <S:faultstring>Client Error</S:faultstring>
         <S:faultactor>soap://C.com</S:faultactor>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

WS-Routing uses a fault code numbering scheme similar to that seen in HTTP but the specific values are completely orthogonal to any status codes defined by HTTP and they are only used to indicate faults. The way SOAP defines fault codes avoids conflicts between fault codes defined in a decentralized environment. The WS-Routing fault codes are therefore orthogonal to fault codes defined by other SOAP-based protocols.

WS-Routing defines the following two classes of fault codes:

7xx WS-Routing Sender fault (see section 5.2.1)
: The WS-Routing sender sent what appears to be a faulty WS-Routing header that cannot be fulfilled because of insufficient or unacceptable information.

8xx WS-Routing Receiver fault (see section 5.2.2)
: The WS-Routing receiver failed to fulfill an apparently valid WS-Routing header because of a fault situation not directly attributable to the received header.

The "xx" part of the WS-Routing fault codes is not significant. In the next two sections, the specific WS-Routing fault codes and their associated reason phrases are defined.

### 5.2.1 WS-Routing Sender Faults

The WS-Routing sender sent what appears to be a faulty WS-Routing header that cannot be fulfilled because of insufficient or unacceptable information. Unless explicitly stated, no information is given to whether the fault situation is permanent or temporary.

#### 5.2.1.1 700 Invalid WS-Routing Header

The WS-Routing header is either malformed or incomplete. No further information is given about what caused the fault. The 700 fault code is special in that a recipient of a WS-Routing fault message SHOULD treat unknown 7xx fault codes as 700 "Invalid WS-Routing Header". The message SHOULD NOT be retried without modifications.

#### 5.2.1.2 701 WS-Routing Header Required

A WS-Routing header entry was not present in the SOAP message and is therefore not compliant with WS-Routing (see section 2.1). This fault code is appropriate if there is no WS-Routing header present at all within the message resulting in the fault. If there is a WS-Routing header present and it is malformed or incomplete then a 700 "Invalid WS-Routing Header" fault code is appropriate. The message SHOULD NOT be retried without modifications.

#### 5.2.1.3 710 Endpoint Not Found

An endpoint in the message path was not found. This fault code is appropriate if the WS-Routing receiver would otherwise have been able to service that endpoint (see also the 712 "Endpoint Not Supported" fault code). The endpoint can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing receiver MUST indicate the endpoint in question in the "endpoint" fault sub-element.

An example of a 710 WS-Routing fault element (XML Namespace prefix left out for illustrative reasons)

```
<fault>
  <code>710</code>
  <reason>Endpoint Not Found</reason>
  <endpoint>soap://D.com/some/endpoint</endpoint>
</fault>
```

**From http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp**

#### 5.2.1.4 711 Endpoint Gone

An endpoint in the message path is known to not exist anymore and no alternative endpoint is known. This fault code is appropriate if the WS-Routing receiver knows that the endpoint used to exist but has been permanently removed. The fault code is also appropriate if the implicit message path indicated by an empty "via" element is no longer available, for example because the underlying communication channel has ceased to exist. The endpoint can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing receiver MUST indicate the endpoint in question in the "endpoint" fault sub-element.

#### 5.2.1.5 712 Endpoint Not Supported

An endpoint in the message path is not supported by the WS-Routing receiver. This fault code is appropriate if the URI contains a URI scheme that the WS-Routing receiver does not support or if the URI points to a part of the URI space that the WS-Routing receiver does not service. The endpoint can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing receiver MUST indicate the endpoint in question in the "endpoint" fault sub-element.

#### 5.2.1.6 713 Endpoint Invalid

An endpoint in the message path does either not follow the URI syntax defined by RFC 2396 for absolute URIs, is not an absolute URI, or contains a fragment identifier (see section 5.3). The endpoint can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing receiver MUST indicate the endpoint in question in the "endpoint" fault sub-element. The message SHOULD NOT be retried without modifications.

#### 5.2.1.7 720 Alternative Endpoint Found

One or more alternative endpoints were found for the requested service. This fault code is appropriate if the WS-Routing receiver knows that the service is available using any of the alternative endpoints rather than the endpoint indicated in the received message. This is analogous to an HTTP redirection.

The endpoint for which alternative endpoints were found can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing receiver MUST indicate the endpoint for which one or more alternative endpoints were found in the "endpoint" fault sub-element as well as provide the alternative endpoint or endpoints within the "found" fault sub-element.

An example of a 720 WS-Routing fault element (XML Namespace prefix left out for illustrative reasons)

```
<fault>
  <code>720</code>
  <reason>Endpoint Moved</reason>
  <endpoint>soap://D.com/some/endpoint</endpoint>
  <found>
    <at>soap://D.com/new/endpoint</at>
    <at>soap://mirror.D.com/new/endpoint</at>
  </found>
</fault>
```

#### 5.2.1.8 730 Endpoint Too Long

The URI used to identify the endpoint is longer than the WS-Routing receiver can handle (see section 5.3). This fault code is appropriate if the size of a URI identifying an endpoint in the WS-Routing header prevents the WS-Routing receiver from interpreting the WS-Routing message. Because of the problem in handling the size of the URI, the endpoint is NOT included in any WS-Routing fault message. The WS-Routing receiver MAY indicate the maximum acceptable size in number of octets using the "maxsize" fault sub-element as illustrated below:

```
<fault>
  <code>730</code>
  <reason>Endpoint Too Long</reason>
  <maxsize>8192</maxsize>
</fault>
```

**From http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp**

#### 5.2.1.9 731 Message Too Large

The size of the overall WS-Routing message prevents the WS-Routing receiver from interpreting the message. There is no suggested size that a WS-Routing receiver must be able to handle as it may depend on the underlying protocol used (see for example section 7.3). The WS-Routing receiver MAY indicate the maximum acceptable size in number of octets using the "maxsize" fault sub-element.

#### 5.2.1.10 740 Message Timeout

The WS-Routing sender did not complete the exchange of the WS-Routing message within the time that the WS-Routing receiver was prepared to wait (see also section 5.4). Upon receipt of this fault code the WS-Routing sender MUST immediately cease transferring the message. The WS-Routing sender MAY repeat the request without modifications at a later time. The WS-Routing receiver MAY indicate the maximum acceptable amount of time in number of seconds in the "maxtime" fault sub-element as illustrated below:

```
<fault>
  <code>740</code>
  <reason>Message Timeout</reason>
  <maxtime>1500</maxtime>
</fault>
```

#### 5.2.1.11 750 Message Loop Detected

A WS-Routing receiver has detected a message loop. The WS-Routing receiver MUST indicate the next endpoint in the message path in the "endpoint" fault sub-element. The message SHOULD NOT be retried without modifications.

#### 5.2.1.12 751 Reverse Path Unavailable

A message with a reverse path (see section 4.3) was received by a WS-Routing intermediary but no reverse message path can be established between the intermediary and the next WS-Routing receiver. This fault code is appropriate if a reverse path is present in the message but the intermediary cannot forward the message in a manner that either explicitly or implicitly defines a reverse message path. Rather than forwarding the message, the intermediary MUST generate a fault to be returned to the initial sender via the reverse path.

#### 5.2.2 WS-Routing Receiver Faults

The WS-Routing receiver failed to fulfill an apparently valid WS-Routing header because of a fault situation not directly attributable to the received message. Unless explicitly stated, no information is given to whether the fault situation is permanent or temporary.

##### 5.2.2.1 800 Unknown WS-Routing Fault

The WS-Routing receiver encountered an unexpected condition that prevented it from handling the received message. The 800 fault code is special in that a recipient of a WS-Routing fault message SHOULD treat unknown 8xx fault codes as 800 (Unknown WS-Routing Fault).

##### 5.2.2.2 810 Element Not Implemented

The WS-Routing header used an element defined in this specification not supported by the WS-Routing receiver. This fault code is only appropriate for elements defined by this specification. WS-Routing header elements not defined by this specification SHOULD be ignored by a WS-Routing receiver and MUST be forwarded by WS-Routing intermediaries.

##### 5.2.2.3 811 Service Unavailable

The WS-Routing receiver is unable to handle the incoming WS-Routing message due to known internal problems or maintenance of the receiver. An example of an internal problem can be a service internal to the WS-Routing receiver that is not responding.

If a time is known for when the WS-Routing receiver is expected to be able to service the incoming message, the receiver MAY indicate the amount of time in number of seconds in the "retryAfter" fault sub-element as illustrated below:

```
<fault>
  <code>811</code>
  <reason>Service Unavailable</reason>
  <retryAfter>1100</retryAfter>
</fault>
```

The WS-Routing sender MAY resend the message after the time interval indicated in the "retryAfter" sub-element has passed. If no value is given, no statement is made as to whether the situation is temporary or not. In that case, the WS-Routing sender SHOULD NOT resend the message until at least 5 minutes has passed.

#### 5.2.2.4 812 Service Too Busy

The WS-Routing receiver is too busy to handle the incoming WS-Routing message. If the WS-Routing receiver does not wish to indicate that it is too busy, it MAY use an 811 "Service Unavailable" fault code instead, or it MAY completely refuse to read the incoming message off the network.

If a time is known for when the WS-Routing receiver is expected to be less busy, the WS-Routing receiver MAY indicate the amount of time in number of seconds in the "retryAfter" fault sub-element. If no "retryAfter" value is given, the WS-Routing sender SHOULD NOT resend the message until at least 5 minutes has passed.

#### 5.2.2.5 820 Endpoint Not Reachable

An endpoint in the message path is not reachable. This fault code is appropriate if the WS-Routing intermediary cannot determine a network route to the endpoint. The endpoint can either be an intermediary identified by a "via" element or it can be the ultimate WS-Routing receiver identified by the "to" element. The WS-Routing intermediary MUST indicate the endpoint in question in the "endpoint" fault sub-element.

### 5.3 Use of URIs in WS-Routing

WS-Routing uses URIs for all identifies without exceptions. To WS-Routing, a URI is simply a formatted string that identifies—via name, location, or any other characteristic—a resource on the Web.

Whereas section 5.1 defined the elements carrying URIs as values, this section defines the rules and conventions governing the URIs as values.

The use of IP addresses in URIs SHOULD be avoided whenever possible (see RFC 1900). However, when used, the literal format for IPv6 addresses in URIs as described by RFC 2732 [17] SHOULD be supported.

All URIs used in WS-Routing MUST be absolute and MUST NOT include a fragment identifier. A WS-Routing receiver receiving a message containing a relative URI or a URI with a fragment MUST generate a 713 "Endpoint Invalid" fault (see section 5.2).

If a URI used to identify a WS-Routing endpoint in the WS-Routing header is used as a value in a SOAP "actor" header attribute, then the endpoint MUST appropriately deal with the actor attribute as defined by the SOAP/1.1 specification (see [15] section 4.2.2).

WS-Routing does not define any equivalence rules for URIs in general as these are defined by the individual URI schemes and by RFC 2396 (see section 6.0). However, because of inconsistencies with respect to URI equivalence rules in many current URI parsers, it is STRONGLY RECOMMENDED that WS-Routing senders do not rely on any equivalence rules in WS-Routing receivers in order to determine equivalence between URI values used in a WS-Routing message.

WS-Routing does not place any a priori limit on the length of a URI. Any WS-Routing receiver MUST be able to handle the length of any URI that it publishes and both WS-Routing senders and WS-Routing receivers SHOULD be able to deal with URIs of at least 8k in length. A WS-Routing receiver SHOULD generate a 730 "Endpoint Too Long" fault if a URI is longer than the receiver can handle (see section 5.2).

### 5.4 Timeouts

Other than for WS-Routing fault messages, WS-Routing does not provide a mechanism for indicating in a WS-Routing message how long a WS-Routing sender is willing to maintain a reverse path. It is expected that other SOAP-based protocols will provide guaranteed delivery along with retransmission policies.

WS-Routing senders and receivers MUST implement a default set of timeouts and MUST use the WS-Routing fault codes for dealing with timeouts when they occur (see section 5.2). WS-Routing timeouts SHOULD be on a per single message basis rather than on a per message flow basis.

It is STRONGLY RECOMMENDED that WS-Routing receivers use the following default values for timeouts:

Message exchange initiation: 2 minutes
: This is while awaiting the first chunk of a message.

Subsequent message: 2 minutes
: This is while awaiting each subsequent chunk of the message.

The timeouts SHOULD be reset after the completion of a single message. A WS-Routing receiver SHOULD NOT have a timeout for the overall completion of a message as this may severely affect the exchange of large messages.

## 6. The soap: URI Scheme

> **Editor Note** This is a preliminary description of the soap: URI scheme. The scheme has not been registered according to the IETF rules for new URI schemes. It may change as part of the registration process.

The "soap:" URI scheme identifies a hierarchical URI namespace that can be used to identify resources on the Web. The default protocol used for retrieving "soap:" URIs is WS-Routing. Other protocols may be possible although this specification does not describe such protocols.

This section defines the scheme-specific syntax and semantics for "soap:" URIs.

```
soap_URI = "soap:" "//" host [ ":" port ] [ abs_path [ ";" up ] [ "?" query ]]

up = "up" "=" ( "tcp" | "udp" )
```

The "port", "abs_path", and "query" parameters are defined by RFC 2396 [4]. The "host" parameter is defined by RFC 2732 [17] in order to support literal IPv6 addresses (see also RFC 1900 for use of literal IP addresses in URIs).

The "up" (underlying protocol) parameter MUST NOT include any LWS. If no "up" parameter is given, TCP is assumed (see section 7). If the port is empty or not provided, port XX is assumed.

Examples of "soap:" URIs:

```
soap://w3.org/2001/02/registration

soap://oneway.org/news/update;up=udp

soap://stock.org/stock/ticker?symbol=msft

soap://oneway.org/finance/stock/ticker;up=udp?symbol=msft
```

### 6.1 Equivalence Rules

The "soap:" URI scheme follows strictly the syntax and equivalence rules defined by RFC 2396 [4] with the following clarifications:

- The "soap:" scheme name is case *insensitive*
- The "host" value is case *insensitive*
- A "soap:" URI using an explicit port number where the port number is XX is equivalent to an "soap:" URI without an explicit port number
- An empty "abs_path" value is equivalent to an "abs_path" value of "/"
- Characters other than those in the "reserved" and "unsafe" sets (see RFC 2396 [4]) are equivalent to their ""%" HEX HEX" encoding

The "up" parameter is always ignored when considering equivalence between soap: URIs. For example, the following three soap: URIs are equivalent in the sense that they refer to the same resource:

```
soap://some.org/foo

soap://some.org/foo;up=udp

soap://some.org/foo;up=tcp
```

### 7. WS-Routing Underlying Protocol Bindings

WS-Routing provides bindings for TCP, UDP, and HTTP as underlying protocols but other bindings are possible. WS-Routing does not define retransmission policies in order to compensate for the different level of service provided by the various underlying protocols. This is expected to be defined by other SOAP-based protocols.

### 7.1 WS-Routing and DIME

Both the TCP binding and the UDP binding requires DIME [18] as the encapsulation mechanism for WS-Routing messages. DIME MAY be used for the HTTP protocol binding but this is not a requirement.

DIME is a lightweight, binary encapsulation format that can be used to encapsulate multiple application defined entities or payloads of arbitrary type and size into a single message construct. It provides the functionality needed for carrying non-XML data along with WS-Routing messages as well as provides an efficient mechanism for determining the boundaries between WS-Routing messages.

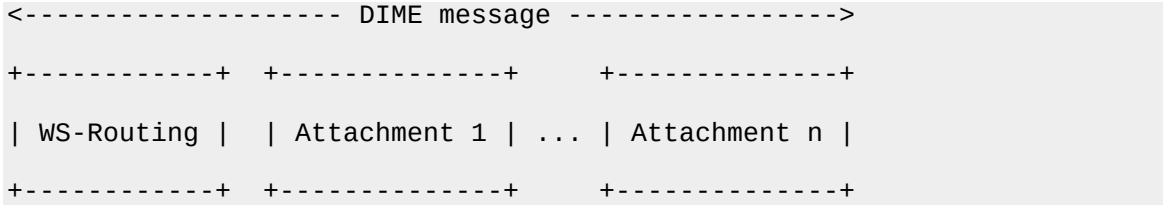


**Figure 3** Example of a WS-Routing message encapsulated in DIME. The message moves in the direction from right to left. The first DIME record is a WS-Routing message and the remaining records can be used for carrying attachments. If there are no attachments then the WS-Routing message is the only record in the DIME message.

The binding between WS-Routing and DIME is defined as follows:

- The first DIME record in a DIME message MUST be a WS-Routing message. The DIME payload identifier of the first WS-Routing message MUST be the **next** WS-Routing receiver in the WS-Routing message path.
- Any subsequent records in a DIME message MUST be treated as attachments. DIME payload identifiers can be used to refer to each attachment from within the WS-Routing message or between attachments.
- The DIME payload type (see [18] section 4.2.2) for a WS-Routing message is the absolute URI "http://www.xmlsoap.org/rp".

**Table 1** Example of what the DIME message would look for the WS-Routing message illustrated in Example 3 without attachments and without chunking the WS-Routing message over multiple DIME records.

| DIME Field | DIME Field Value |
|---|---|
| MB | 1 |
| ME | 1 |
| CF | 0 |
| TNF | 2 |
| TYPE | http://schemas.xmlsoap.org/rp/ |
| ID | soap://C.com |
| DATA | <S:Envelope... |

### 7.2 TCP

**Editor Note** The TCP binding described in this section is preliminary and is likely to change before requesting a well-known port from IANA.

WS-Routing over TCP MUST use DIME as the encapsulation mechanism (see section 7.2). The implicit reverse message path defined for TCP is the source address of the peer at the other end of the TCP connection independent of who initiated the connection.

The following rules apply independently of who initiated the TCP connection:

- Any WS-Routing message can be sent in either direction on a single TCP connection independent of whether it is a reply, a fault, or any other message.
- There is no ordering requirement between WS-Routing messages allowing messages and replies to be sent out of order in either direction
- WS-Routing messages can be sent back-to-back allowing multiple messages to be exchanged in serial along both message paths at any given time

By using explicit "via" element values in the reverse path, messages traveling along the reverse path do not have to be sent on the same TCP connection as the forward path messages.

### 7.2.1 TCP Connection Management

Implementations SHOULD reuse existing TCP connections wherever possible. It is STRONGLY RECOMMENDED that WS-Routing uses a timeout based mechanism for determining when an idle TCP connection should be closed and that the default timeout for an idle TCP connection is 2 minutes (see also section 5.4).

When a WS-Routing sender or receiver wishes to time out a TCP connection it SHOULD issue a graceful close on the outgoing connection and allow the incoming connection to drain. In many TCP socket APIs this can be done using a *shutdown* system call. WS-Routing senders and receivers SHOULD both watch for the other side of the TCP connection close the connection, and respond to it as appropriate.

If a TCP connection is closed and there are outstanding messages that were to be sent on the implicit reverse path, then these messages SHOULD be discarded.

WS-Routing senders SHOULD limit the number of simultaneous connections that they maintain to a given WS-Routing receiver. An initial WS-Routing sender SHOULD NOT maintain more than 2 TCP connections with any WS-Routing receiver. A WS-Routing intermediary MAY use up to 2*N TCP connections with any WS-Routing receiver, where N is the number of WS-Routing senders using that intermediary.

WS-Routing intermediaries MAY reuse the same TCP connection for forwarding WS-Routing messages even though these messages may have arrived on different TCP connections. There is no requirement that there be a 1:1 mapping of incoming and outgoing TCP connections. An example of how a TCP connection may be reused is illustrated in Figure 3:

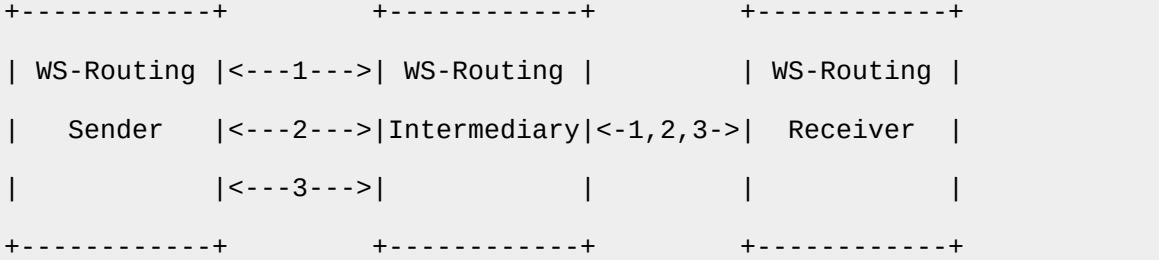


**Figure 3** An example of possible mapping between TCP connections in a WS-Routing intermediary

### 7.3 UDP

**Editor Note** The UDP binding described in this section is preliminary and is likely to change before requesting a well-known port from IANA.

WS-Routing over UDP MUST use DIME as the encapsulation mechanism (see section 7.2). There is no implicit reverse message path defined for UDP. This means that if a reverse path is requested then it MUST be explicitly indicated using a non-empty "via" element in the "rev" element (see section 4.3).

WS-Routing uses the same rule of thumb as SIP [5], which means that UDP datagrams, including all headers, SHOULD NOT be larger than the path maximum transmission unit (MTU) if the MTU is known, or 1500 bytes if the MTU is unknown. Furthermore, SIP provides the following recommendation, which also applies to WS-Routing:

The 1500 bytes accommodates encapsulation within the "typical" ethernet MTU without IP fragmentation. Recent studies [12] indicate that an MTU of 1500 bytes is a reasonable assumption. The next lower common MTU values are 1006 bytes for SLIP and 296 for low-delay PPP (RFC 1191 [1]). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

WS-Routing does not define any reliability or retransmission policies for UDP, as these are expected to be specified by other SOAP reliable messaging mechanisms.

### 7.4 HTTP

As mentioned in section 3.4, it is possible to carry WS-Routing messages using a variety of underlying protocols. Although HTTP is not a transport like TCP and UDP, it supports the notion of carrying a SOAP message within the framework of HTTP and can act as the underlying protocol for a part of the WS-Routing message path. The SOAP specification (see [15] section 6) defines the rules for carrying a SOAP message within an HTTP message. These rules apply equally to WS-Routing but in addition, this section defines a set of rules that specifically applies to carrying WS-Routing messages within HTTP.

When a WS-Routing message is carried within an HTTP message there is no mechanism for guaranteeing that there is a WS-Routing receiver at the destination of the HTTP message. In order to ensure that a SOAP processor honors the WS-Routing semantics, a WS-Routing sender sending a WS-Routing message using HTTP SHOULD mark the WS-Routing header with a SOAP mustUnderstand attribute with a value of "1" (see [15] section 4.2.3) and a SOAP actor attribute with a value of "http://schemas.xmlsoap.org/soap/actor/next" (see [15] section 4.2.2)

#### 7.4.1 HTTP Request

When generating an HTTP request message there are two values that are mapped from the WS-Routing message into the HTTP message: the Request-URI and the SOAPAction header field (see [15] section 6.1.1). A WS-Routing sender generating an HTTP message MUST use the following mappings between the WS-Routing header and the HTTP header:

Request-URI
: The value of the Request-URI SHOULD be the value of the next WS-Routing receiver in the message path regardless of whether it is the ultimate WS-

Routing receiver or a WS-Routing intermediary. That is, if the next WS-Routing receiver is specified by a "via" element in the forward message path, then the value of the Request-URI is same as the value of that "via" element. If the next receiver is specified by the "to" element then the value of the Request-URI is the same as the value of the "to" element (see section 4.4).

SOAPAction Header field

The value of the SOAPAction header MUST be the same as the value of the "action" element (see section 5.1.1).

A WS-Routing message included in an HTTP request message MAY contain a reverse path. The implicit reverse path (indicated by an empty "via" element) for a WS-Routing message included in an HTTP request is the corresponding HTTP response message. Note that as WS-Routing doesn't define a request/response message path pattern, it is up to the application to define whether the implicit reverse path is to be used. The rules for dealing with a reverse path in an HTTP response is described in section 7.2.2.

Below is an example of a WS-Routing message included in an HTTP request. The example shows how HTTP can be a part of the overall WS-Routing message path and illustrates the WS-Routing message being sent from WS-Routing intermediary **B** to intermediary **C** in Figure 1. In the example, **C** is identified by a URI of form "http://..." although this doesn't have to be the case.

**Example 11** Message leaving intermediary B in forward direction towards C using HTTP POST /endpoint/on/http/server HTTP/1.1

```
Host: C.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://www.im.org/chat"

<S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header>
      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/"
```

```
       S:mustUnderstand="1"
S:actor="http://schemas.xmlsoap.org/soap/actor/next">

         <m:action>http://www.im.org/chat</m:action>

         <m:to>soap://D.com/some/endpoint</m:to>

         <m:fwd>

            <m:via>http://C.com/endpoint/on/http/server</m:via>

         </m:fwd>

         <m:rev>

            <m:via/>

            <m:via m:vid="cid:122326@B.com"/>

         </m:rev>

         <m:from>mailto:henrikn@microsoft.com</m:from>

         <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>

      </m:path>

   </S:Header>

   <S:Body>

      ...

   </S:Body>

</S:Envelope>
```

### 7.4.2 HTTP Response

Any WS-Routing message can be carried within an HTTP response message. There is no mapping between any HTTP header fields in an HTTP response and the values of the WS-Routing header like there is for a WS-Routing message in an HTTP request message.

A WS-Routing message included in an HTTP response message MAY contain a reverse path. There is no implicit reverse path (indicated by an empty "via" element) defined for a WS-Routing message included in an HTTP response so the "via" element describing the reverse path MUST be explicit.

These are the rules for choosing the appropriate HTTP status code for an HTTP response that is generated in response to an HTTP request message containing a WS-Routing message:

- If an *implicit* reverse path is present in the WS-Routing message in the HTTP request and the request is successfully fulfilled but no WS-Routing message is to be sent along the reverse path then a 204 (No Content) or a 200 (Ok) status code is the appropriate HTTP status code to use.
- If an *explicit* reverse path is present in the WS-Routing message in the HTTP request and the request is successfully fulfilled then a 204 (No Content) or a 200 (Ok) status code is the appropriate HTTP status code to use.
- If *no* reverse path is present and the request is successfully fulfilled then a 204 (No Content) or a 200 (Ok) status code is the appropriate HTTP status code to use.
- In either of the three cases above, if the HTTP request has been accepted but not processed then a 202 (Accepted) HTTP status code is the appropriate code to use.
- As for all SOAP fault messages carried in an HTTP response message, a 500 (Internal Server Error) is the appropriate HTTP status code to use.

Below is an example of an HTTP response message carrying a message from WS-Routing intermediary **C** to intermediary **B** in Figure 1. Note the non-empty "via" element indicating that the explicit reverse path for the message. The reverse path is explicit because the HTTP binding doesn't define an implicit reverse path.

**From http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-routing.asp**

**Example 12** Message leaving intermediary C in reverse direction towards B using HTTP

```
HTTP/1.1 200 Ok

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn


<S:Envelope

   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">

   <S:Header>

      <m:path xmlns:m="http://schemas.xmlsoap.org/rp/"

       S:mustUnderstand="1"
S:actor="http://schemas.xmlsoap.org/soap/actor/next">

         <m:action>http://www.im.org/update</m:action>

         <m:fwd>

            <m:via/>

            <m:via m:vid="cid:122326@B.com"/>

         </m:fwd>

         <m:rev>

            <m:via>http://C.com/endpoint/on/http/server</m:via>

            <m:via>soap://D.com/some/endpoint</m:via>

         </m:rev>

         <m:from>mailto:satisht@microsoft.com</m:from>

         <m:id>uuid:9fshs8fj-sffg-r5ts-adfg-9kd84jd9mjdld43</m:id>

         <m:relatesTo>uuid:84b9f5d0-33fb-4a81-b02b-
5b760641c1d6</m:relatesTo>

      </m:path>

   </S:Header>

   <S:Body>

      ...

   </S:Body>
```

```
</S:Envelope>
```

## 8. Acknowledgements

TBD

## 9. References

**[1]** Mogul, J. and S. Deering, "Path MTU Discovery", RFC1191, November 1990.

**[2]** S. Bradner, "The Internet Standards Process -- Revision 3", RFC2026, Harvard University, October 1996

**[3]** S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997

**[4]** T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

**[5]** M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, "SIP: Session Initiation Protocol", RFC2543, ACIRI, Columbia U., Cal Tech, Bell Labs, March 1999

**[6]** R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, U.C. Irvine, DEC W3C/MIT, DEC, W3C/MIT, W3C/MIT, January 1997

**[7]** W3C Recommendation "The XML Specification"

**[8]** W3C Recommendation "Namespaces in XML"

**[9]** W3C Proposed Recommendation "XML Linking Language"

**[10]** W3C Recommendation "XML Schema Part 1: Structures"

**[11]** W3C Recommendation "XML Schema Part 2: Datatypes"

**[12]** Stevens, W., "TCP/IP illustrated: the protocols , vol. 1". Reading, Massachusetts: Addison-Wesley, 1994.

**[13]** Rohit Khare, "Composing Active Proxies to Extend the Web"

**[14]** H. F. Nielsen, P. Leach, S. Lawrence, "An HTTP Extension Framework", RFC 2774, Microsoft, Agranat Systems, Feb 2000

**[15]** W3C Note "Simple Object Access Protocol (SOAP) 1.1"

**[16]** W3C Note "SOAP Messages with Attachments"

**[17]** R. Hinden, B. Carpenter, L. Masinter, "Format for Literal IPv6 Addresses in URL's", RFC 2732, Nokia, IBM, AT&T, December 1999

**[18]** H. Nielsen, H. Sanders, E. Christensen, "DIME – Direct Internet Message Encapsulation", Microsoft, May 2001