SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)

W3C Recommendation 27 April 2007

This version:
http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
Latest version:
http://www.w3.org/TR/soap12-part1/
Previous versions:
http://www.w3.org/TR/2006/PER-soap12-part1-20061219/
Editors:
Martin Gudgin, Microsoft
Marc Hadley, Sun Microsystems
Noah Mendelsohn, IBM
Jean-Jacques Moreau, Canon
Henrik Frystyk Nielsen, Microsoft
Anish Karmarkar, Oracle
Yves Lafon, W3C

Please refer to the **errata** for this document, which may include normative corrections.

See also translations.

Abstract

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. "Part 1: Messaging Framework" defines, using XML technologies, an extensible messaging framework containing a message construct that can be exchanged over a variety of underlying protocols.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a W3C Recommendation. It has been produced by the XML Protocol Working Group, which is part of the Web Services Activity. This second edition updates and supersedes the original Recommendation by the inclusion of the accumulated errata. Changes between these two versions are described in a diff document.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Please report errors in this document to the public mailing list xmlp-comments@w3.org (archive). It is inappropriate to send discussion email to this address.

SOAP Version 1.2 supercedes all previous versions of SOAP, including SOAP Version 1.1 [SOAP 1.1]

The SOAP 1.2 Implementation Report can be found at http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html.

This document is governed by the 24 January 2002 CPP as amended by the W3C Patent Policy Transition Procedure. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

A list of current W3C Recommendations and other technical reports can be found at http://www.w3.org/TR.

Short Table of Contents

1. Introduction
2. SOAP Processing Model
3. SOAP Extensibility Model
4. SOAP Protocol Binding Framework
5. SOAP Message Construct
6. Use of URIs in SOAP
7. Security Considerations
8. References
A. Version Transition From SOAP/1.1 to SOAP Version 1.2
B. Acknowledgements (Non-Normative)

Table of Contents

1. Introduction
   1.1 Notational Conventions
   1.2 Conformance
   1.3 Relation to Other Specifications
      1.3.1 Processing Requirements

1.4 Example SOAP Message
1.5 SOAP Terminology
1.5.1 Protocol Concepts
1.5.2 Data Encapsulation Concepts
1.5.3 Message Sender and Receiver Concepts
2. SOAP Processing Model
2.1 SOAP Nodes
2.2 SOAP Roles and SOAP Nodes
2.3 Targeting SOAP Header Blocks
2.4 Understanding SOAP Header Blocks
2.5 Structure and Interpretation of SOAP Bodies
2.6 Processing SOAP Messages
2.7 Relaying SOAP Messages
2.7.1 Relaying SOAP Header Blocks
2.7.2 SOAP Forwarding Intermediaries
2.7.2.1 Relayed Infoset
2.7.3 SOAP Active Intermediaries
2.8 SOAP Versioning Model
3. SOAP Extensibility Model
3.1 SOAP Features
3.1.1 Requirements on Features
3.2 SOAP Message Exchange Patterns (MEPs)
3.3 SOAP Modules
4. SOAP Protocol Binding Framework
4.1 Goals of the Binding Framework
4.2 Binding Framework
5. SOAP Message Construct
5.1 SOAP Envelope
5.1.1 SOAP encodingStyle Attribute
5.2 SOAP Header
5.2.1 SOAP header block
5.2.2 SOAP role Attribute
5.2.3 SOAP mustUnderstand Attribute
5.2.4 SOAP relay Attribute
5.3 SOAP Body
5.3.1 SOAP Body child Element
5.4 SOAP Fault
5.4.1 SOAP Code Element
5.4.1.1 SOAP Value element (with Code parent)
5.4.1.2 SOAP Subcode element
5.4.1.3 SOAP Value element (with Subcode parent)
5.4.2 SOAP Reason Element
5.4.2.1 SOAP Text Element
5.4.3 SOAP Node Element
5.4.4 SOAP Role Element
5.4.5 SOAP Detail Element
5.4.5.1 SOAP detail entry
5.4.6 SOAP Fault Codes
5.4.7 VersionMismatch Faults
5.4.7.1 SOAP Upgrade Header Block
5.4.7.2 SOAP SupportedEnvelope Element
5.4.7.3 SOAP QName Attribute
5.4.7.4 VersionMismatch Example
5.4.8 SOAP mustUnderstand Faults

5.4.8.1 SOAP NotUnderstood Element
5.4.8.2 SOAP QName Attribute
5.4.8.3 NotUnderstood Example
6. Use of URIs in SOAP
7. Security Considerations
7.1 SOAP Nodes
7.2 SOAP Intermediaries
7.3 Underlying Protocol Bindings
7.3.1 Binding to Application-Specific Protocols
8. References
8.1 Normative References
8.2 Informative References

Appendices

A. Version Transition From SOAP/1.1 to SOAP Version 1.2
B. Acknowledgements (Non-Normative)

---

1. Introduction

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

Two major design goals for SOAP are simplicity and extensibility (see XMLP Requirements [XMLP Requirements]). SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often found in distributed systems. Such features include but are not limited to "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs). While it is anticipated that many features will be defined, this specification provides specifics only for two MEPs. Other features are left to be defined as extensions by other specifications.

The SOAP Version 1.2 specification consists of three parts. Part 1 of the SOAP Version 1.2 specification (this document) defines the SOAP messaging framework consisting of:

1. The SOAP processing model defining the rules for processing a SOAP message (see **2. SOAP Processing Model**).
2. The SOAP Extensibility model defining the concepts of SOAP features and SOAP modules (see **3. SOAP Extensibility Model**).
3. The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes (see **4. SOAP Protocol Binding Framework**).
4. The SOAP message construct defining the structure of a SOAP message (see **5. SOAP Message Construct**).

The SOAP 1.2 Primer [SOAP Part 0] is a non-normative document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications.

SOAP 1.2 Part 2 [SOAP Part 2] describes a set of adjuncts that can be used in connection with the SOAP messaging framework.

Note:

In previous versions of this specification the SOAP name was an acronym. This is no longer the case.

1.1 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 [RFC 2119].

This specification uses a number of namespace prefixes throughout; they are listed in **Table 1**. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see XML Infoset [XML InfoSet]).

Table 1: Prefixes and Namespaces used in this specification.

Prefix	Namespace	Notes
env	"http://www.w3.org/2003/05/soap-envelope"	A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-envelope" namespace can be found at http://www.w3.org/2003/05/soap-envelope.
xs	"http://www.w3.org/2001/XMLSchema"	The namespace of the XML Schema [XML Schema Part 1], [XML Schema Part 2] specification

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs (see RFC 3986 [RFC 3986]).

All parts of this specification are normative, with the exception of examples and sections explicitly marked as "Non-Normative".

1.2 Conformance

This specification describes data formats, and the rules for generating, exchanging, and processing messages using those formats. This specification does not mandate the scope of any particular implementation, although it requires that no implementation violates any mandatory requirement.

For an implementation to claim conformance with the SOAP Version 1.2 specification, it MUST correctly implement all mandatory ("MUST") requirements expressed in Part 1 of the SOAP Version 1.2 specification (this document) that pertain to the activity being performed. Note that an implementation is not mandated to implement all the mandatory requirements. For example, a special purpose implementation that never sends a SOAP header block can claim conformance provided that it correctly implements the mandatory requirements that pertain to the messages it does send.

An implementation MAY implement any number of the Adjuncts specified in SOAP 1.2 Part 2 [SOAP Part 2]. Note that no conformance is associated with the convention for describing features and bindings (see **3. SOAP Extensibility Model** and **4. SOAP Protocol Binding Framework**). The implementation of an Adjunct MUST implement all the pertinent mandatory requirements expressed in the specification of the Adjunct to claim conformance with the Adjunct.

SOAP Version 1.2 can be used as the basis for other technologies that provide richer or more specialized services. To claim conformance with the SOAP Version 1.2 specification, the specifications and implementations of such technologies must be consistent with the pertinent mandatory requirements expressed in Part 1 of the SOAP Version 1.2 specification (this document). Rules for conformance with such new specifications are beyond the scope of the SOAP Version 1.2 specification; it is recommended that specifications for such technologies provide the appropriate conformance rules.

SOAP Version 1.2 is designed to enable at least the usage scenarios described in SOAP 1.2 Usage Scenarios [SOAP Usage Scenarios], and possibly other scenarios. Informal descriptions showing XML representations of concrete SOAP messages used in some common scenarios are provided in SOAP 1.2 Part 0 [SOAP Part 0].

1.3 Relation to Other Specifications

A SOAP message is specified as an XML Information Set [XML InfoSet]. While all SOAP message examples in this document are shown using XML 1.0 [XML 1.0] syntax, other representations MAY be used to transmit SOAP messages between nodes (see **4. SOAP Protocol Binding Framework**).

Some of the *information items* defined by this document (see **5. SOAP Message Construct**) are identified using namespace-qualified names [Namespaces in XML]. See **Table 1** for a list of the namespace names defined in this document.

> **Note:**
>
> This specification uses the term *XML Expanded Name* to refer to the value space pair {absolute uri reference,local-name} for a value of type xsd:QName. Similar terminology is under consideration for inclusion in future versions of Namespace in XML [Namespaces in XML]. Should future versions of namespace in XML [Namespaces in XML] adopt alternative terminology, we anticipate that corresponding changes will be made to this recommendation in the form of an erratum, or in conjunction with some other future revision.

SOAP does not require that XML Schema processing (assessment or validation) be performed to establish the correctness or 'schema implied' values of *element* and *attribute information items* defined by Parts 1 and 2 of this specification. The values associated with *element* and *attribute information items* defined in this specification MUST be carried explicitly in the transmitted SOAP message except where stated otherwise (see **5. SOAP Message Construct**).

SOAP *attribute information items* have types described by XML Schema [XML Schema Part 2]. Unless otherwise stated, all lexical forms are supported for each such attribute, and lexical forms representing the same value in the XML Schema value space are considered equivalent for purposes of SOAP processing, e.g., the boolean lexical forms "1" and "true" are interchangeable. For brevity, text in this

specification refers only to one lexical form for each value, e.g., "if the value of the `mustUnderstand` *attribute information item* is 'true'".

Specifications for the processing of application-defined data carried in a SOAP message but not defined by this specification MAY call for additional validation of the SOAP message in conjunction with application-level processing. In such cases, the choice of schema language and/or validation technology is at the discretion of the application.

SOAP uses XML Base [XML Base] for determining a base URI for relative URI references used as values in *information items* defined by this specification (see **6. Use of URIs in SOAP**).

The media type "application/soap+xml" SHOULD be used for XML 1.0 serializations of the SOAP message infoset (see SOAP 1.2 Part 2 [SOAP Part 2], The "application/soap+xml" Media Type).

1.3.1 Processing Requirements

The ability to use SOAP in a particular environment will vary depending on the actual constraints, choice of tools, processing model, or nature of the messages being exchanged. SOAP has been designed to have a relatively small number of dependencies on other XML specifications, none of which are perceived as having prohibitive processing requirements. Also, limiting use of SOAP to small messages instead of arbitrarily-sized messages and supporting only a few specific message types instead of implementing generalized processing could significantly lower processing requirements.

1.4 Example SOAP Message

The following example shows a sample notification message expressed in SOAP. The message contains two pieces of application-defined data not defined by this specification: a SOAP header block with a local name of `alertcontrol` and a body element with a local name of `alert` . In general, SOAP header blocks contain information which might be of use to SOAP intermediaries as well as the ultimate destination of the message. In this example an intermediary might prioritize the delivery of the message based on the priority and expiration information in the SOAP header block. The body contains the actual message payload, in this case the alert message.

Example 1: SOAP message containing a SOAP header block and a SOAP body

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

1.5 SOAP Terminology

This section describes the terms and concepts introduced in Part 1 of the SOAP Version 1.2 specification (this document).

1.5.1 Protocol Concepts

SOAP

The formal set of conventions governing the format and processing rules of a SOAP message. These conventions include the interactions among SOAP nodes generating and accepting SOAP messages for the purpose of exchanging information along a SOAP message path.

SOAP node

The embodiment of the processing logic necessary to transmit, receive, process and/or relay a SOAP message, according to the set of conventions defined by this recommendation. A SOAP node is responsible for enforcing the rules that govern the exchange of SOAP messages (see **2. SOAP Processing Model**). It accesses the services provided by the underlying protocols through one or more SOAP bindings.

SOAP role

A SOAP receiver's expected function in processing a message. A SOAP receiver can act in multiple roles.

SOAP binding

The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange (see **4. SOAP Protocol Binding Framework**). Examples of SOAP bindings include carrying a SOAP message within an HTTP entity-body, or over a TCP stream.

SOAP feature

An extension of the SOAP messaging framework (see **3. SOAP Extensibility Model**). Examples of features include "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs).

SOAP module

A SOAP Module is a specification that contains the combined syntax and semantics of SOAP header blocks specified according to the rules in **3.3 SOAP Modules**. A SOAP module realizes zero or more SOAP features.

SOAP message exchange pattern (MEP)

A template for the exchange of SOAP messages between SOAP nodes enabled by one or more underlying SOAP protocol bindings (see **4. SOAP Protocol Binding Framework**). A SOAP MEP is an example of a SOAP feature (see **3.2 SOAP Message Exchange Patterns (MEPs)**).

SOAP application

An entity, typically software, that produces, consumes or otherwise acts upon

SOAP messages in a manner conforming to the SOAP processing model (see **2. SOAP Processing Model**).

1.5.2 Data Encapsulation Concepts

SOAP message

The basic unit of communication between SOAP nodes.

SOAP envelope

The outermost *element information item* of a SOAP message.

SOAP header

A collection of zero or more SOAP header blocks each of which might be targeted at any SOAP receiver within the SOAP message path.

SOAP header block

An *element information item* used to delimit data that logically constitutes a single computational unit within the SOAP header. The type of a SOAP header block is identified by the XML expanded name of the header block *element information item*.

SOAP body

A collection of zero or more *element information items* targeted at an ultimate SOAP receiver in the SOAP message path (see **5.3 SOAP Body**).

SOAP fault

A SOAP *element information item* which contains fault information generated by a SOAP node.

1.5.3 Message Sender and Receiver Concepts

SOAP sender

A SOAP node that transmits a SOAP message.

SOAP receiver

A SOAP node that accepts a SOAP message.

SOAP message path

The set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver.

Initial SOAP sender

The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.

SOAP intermediary

A SOAP intermediary is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

Ultimate SOAP receiver

The SOAP receiver that is a final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example because of a problem at a SOAP intermediary. An ultimate SOAP receiver cannot also be a SOAP intermediary for the same SOAP message (see **2. SOAP Processing Model**).

2. SOAP Processing Model

SOAP provides a distributed processing model that assumes a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. Note that the SOAP distributed processing model can support many MEPs including but not limited to one-way messages, request/response interactions, and peer-to-peer conversations (see **3.2 SOAP Message Exchange Patterns (MEPs)** for a description of the relationship between SOAP message exchange patterns and the SOAP extensibility model).

This section defines the SOAP distributed processing model. The SOAP processing model specifies how a SOAP receiver processes a SOAP message. It applies to a single message only, in isolation from any other SOAP message. The SOAP processing model itself does not maintain any state or perform any correlation or coordination between messages, even, for example, when used in combination with a SOAP feature which involves sending multiple SOAP messages in sequence, each subsequent message depending on the response to the previous message. It is the responsibility of each such feature to define any combined processing.

Section **3. SOAP Extensibility Model** describes how SOAP can be extended and how SOAP extensions might interact with the SOAP processing model and the SOAP protocol binding framework. Section **4. SOAP Protocol Binding Framework** defines a framework for describing the rules for how SOAP messages can be exchanged over a variety of underlying protocols.

2.1 SOAP Nodes

A SOAP node can be the initial SOAP sender, an ultimate SOAP receiver, or a SOAP intermediary. A SOAP node receiving a SOAP message MUST perform processing according to the SOAP processing model as described in this section and in the remainder of this specification. A SOAP node is identified by a URI, see **5.4.3 SOAP Node Element**

2.2 SOAP Roles and SOAP Nodes

In processing a SOAP message, a SOAP node is said to act in one or more SOAP roles, each of which is identified by a URI known as the SOAP role name. The roles assumed by a node MUST be invariant during the processing of an individual SOAP

message. This specification deals only with the processing of individual SOAP messages. No statement is made regarding the possibility that a given SOAP node might or might not act in varying roles when processing more than one SOAP message.

Table 2 defines three role names which have special significance in a SOAP message (see **2.6 Processing SOAP Messages**).

Table 2: SOAP Roles defined by this specification

Short-name	Name	Description
`next`	"http://www.w3.org/2003/05/soap-envelope/role/next"	Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
`none`	"http://www.w3.org/2003/05/soap-envelope/role/none"	SOAP nodes MUST NOT act in this role.
`ultimateReceiver`	"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"	The ultimate receiver MUST act in this role.

In addition to the SOAP role names defined in **Table 2**, other role names MAY be used as necessary to meet the needs of SOAP applications.

While the purpose of a SOAP role name is to identify a SOAP node or nodes, there are no routing or message exchange semantics associated with the SOAP role name. For example, SOAP roles MAY be named with a URI useable to route SOAP messages to an appropriate SOAP node. Conversely, it is also appropriate to use SOAP roles with names that are related more indirectly to message routing (e.g., "http://example.org/banking/anyAccountMgr") or which are unrelated to routing (e.g., a URI meant to identify "all cache management software". For example, a SOAP header block targeted at such a role might be used to carry an indication to any concerned software that the containing SOAP message is idempotent, and can safely be cached and replayed).

With the exception of the three SOAP role names defined in **Table 2**, this specification does not prescribe the criteria by which a given node determines the set of roles in which it acts on a given message. For example, implementations can base this determination on factors including, but not limited to: hard coded choices in the implementation, information provided by the underlying protocol binding (e.g., the URI to which the message was physically delivered), or configuration information provided by users during system installation.

2.3 Targeting SOAP Header Blocks

A SOAP header block MAY carry a `role` *attribute information item* (see **5.2.2 SOAP role Attribute**) that is used to target the header block at SOAP nodes operating in the specified role. This specification refers to the value of the SOAP `role` *attribute information item* as the SOAP role for the corresponding SOAP header block.

A SOAP header block is said to be targeted at a SOAP node if the SOAP role for the header block is the name of a role in which the SOAP node operates. SOAP header blocks targeted at the special role "http://www.w3.org/2003/05/soap-envelope /role/none" are never formally processed. Such SOAP header blocks MAY carry data that is required for processing of other SOAP header blocks. Unless removed by the action of an intermediary (see **2.7 Relaying SOAP Messages**) , such blocks are

relayed with the message to the ultimate receiver (see also **3.3 SOAP Modules**).

2.4 Understanding SOAP Header Blocks

It is likely that specifications for a wide variety of header functions (i.e., SOAP modules) will be developed over time (see **3.3 SOAP Modules**), and that some SOAP nodes might include the software necessary to implement one or more such extensions. A SOAP header block is said to be understood by a SOAP node if the software at that SOAP node has been written to fully conform to and implement the semantics specified for the XML expanded name of the outer-most *element information item* of that header block.

A SOAP header block MAY carry a `mustUnderstand` *attribute information item* (see **5.2.3 SOAP mustUnderstand Attribute**). When the value of such an *attribute information item* is "true", the SOAP header block is said to be mandatory.

Mandatory SOAP header blocks are presumed to somehow modify the semantics of other SOAP header blocks or SOAP body elements. Therefore, for every mandatory SOAP header block targeted to a node, that node MUST either process the header block or not process the SOAP message at all, and instead generate a fault (see **2.6 Processing SOAP Messages** and **5.4 SOAP Fault**). Tagging SOAP header blocks as mandatory thus assures that such modifications will not be silently (and, presumably, erroneously) ignored by a SOAP node to which the header block is targeted.

The `mustUnderstand` *attribute information item* is not intended as a mechanism for detecting errors in routing, misidentification of nodes, failure of a node to serve in its intended role(s), etc. Any of these conditions can result in a failure to even attempt processing of a given SOAP header block from a SOAP envelope. This specification therefore does not require any fault to be generated based on the presence or value of the `mustUnderstand` *attribute information item* on a SOAP header block not targeted at the current processing node. In particular, it is not an error for an ultimate SOAP receiver to receive a message containing a mandatory SOAP header block that is targeted at a role other than the ones assumed by the ultimate SOAP receiver. This is the case, for example, when a SOAP header block has survived erroneously due to a routing or targeting error at a preceding intermediary.

2.5 Structure and Interpretation of SOAP Bodies

An ultimate SOAP receiver MUST correctly process the immediate children of the SOAP body (see **5.3 SOAP Body**). However, with the exception of SOAP faults (see **5.4 SOAP Fault**), Part 1 of this specification (this document) mandates no particular structure or interpretation of these elements, and provides no standard means for specifying the processing to be done.

2.6 Processing SOAP Messages

This section sets out the rules by which SOAP messages are processed. Nothing in this specification prevents the use of optimistic concurrency, roll back, or other techniques that might provide increased flexibility in processing order. Unless otherwise stated, processing of all generated SOAP messages, SOAP faults and application-level side effects MUST be semantically equivalent to performing the following steps separately, and in the order given.

1. Determine the set of roles in which the node is to act. The contents of the SOAP envelope, including any SOAP header blocks and the SOAP body, MAY be inspected in making such determination.

2. Identify all header blocks targeted at the node that are mandatory.

3. If one or more of the SOAP header blocks identified in the preceding step are not understood by the node then generate a single SOAP fault with the `Value` of `Code` set to "env:MustUnderstand" (see **5.4.8 SOAP mustUnderstand Faults**). If such a fault is generated, any further processing MUST NOT be done. Faults relating to the contents of the SOAP body MUST NOT be generated in this step.

   Note:

   Throughout this document, the term "`Value` of `Code` " is used as a shorthand for "value of the `Value` child *element information item* of the `Code` *element information item*" (see **5.4.1 SOAP Code Element**).

4. Process all mandatory SOAP header blocks targeted at the node and, in the case of an ultimate SOAP receiver, the SOAP body. A SOAP node MAY also choose to process non-mandatory SOAP header blocks targeted at it.

5. In the case of a SOAP intermediary, and where the SOAP message exchange pattern and results of processing (e.g., no fault generated) require that the SOAP message be sent further along the SOAP message path, relay the message as described in section **2.7 Relaying SOAP Messages**.

In all cases where a SOAP header block is processed, the SOAP node MUST understand the SOAP header block and MUST do such processing in a manner fully conformant with the specification for that header block. The successful processing of one header block does not guarantee successful processing of another block with the same XML expanded name within the same message: the specification for the header block determines the circumstances in which such processing would result in a fault. An ultimate SOAP receiver MUST process the SOAP body, in a manner consistent with **2.5 Structure and Interpretation of SOAP Bodies**.

Failure is indicated by the generation of a fault (see **5.4 SOAP Fault**). SOAP message processing MAY result in the generation of a SOAP fault; more than one SOAP fault MUST NOT be generated when processing a SOAP message.

A message may contain or result in multiple errors during processing. Except where the order of detection is specifically indicated (as in **2.4 Understanding SOAP Header Blocks**), a SOAP node is at liberty to reflect any single fault from the set of possible faults prescribed for the errors encountered. The selection of a fault need not be predicated on the application of the "MUST", "SHOULD" or "MAY" keywords to the generation of the fault, with the exception that if one or more of the prescribed faults is qualified with the "MUST" keyword, then any one fault from the set of possible faults MUST be generated.

SOAP nodes MAY make reference to any information in the SOAP envelope when processing a SOAP body or SOAP header block. For example, a caching function can cache the entire SOAP message, if desired.

The processing of one or more SOAP header blocks MAY control or determine the

order of processing for other SOAP header blocks and/or the SOAP body. For example, one could create a SOAP header block to force processing of other SOAP header blocks in lexical order. In the absence of such a controlling SOAP header block, the order of header and body processing is at the discretion of the SOAP node. Header blocks MAY be processed in arbitrary order. Header block processing MAY precede, MAY be interleaved with, or MAY follow processing of the SOAP body. For example, processing of a "begin transaction" header block would typically precede body processing, a "logging" function might run concurrently with body processing and a "commit transaction" header block might be honored following completion of all other work.

Note:

The above rules apply to processing at a single node. SOAP extensions can be designed to ensure that SOAP header blocks are processed in an appropriate order, as the message moves along the message path towards the ultimate SOAP receiver. Specifically, such extensions might specify that a fault with a `Value` of `Code` set to "env:Sender" is generated if some SOAP header blocks have inadvertently survived past some intended point in the message path. Such extensions might depend on the presence or value of the `mustUnderstand` *attribute information item* in the surviving SOAP header blocks when determining whether an error has occurred.

2.7 Relaying SOAP Messages

As mentioned earlier in this section, it is assumed that a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. While SOAP does not itself define any routing or forwarding semantics, it is anticipated that such functionality can be described as one or more SOAP features (see **3. SOAP Extensibility Model**). The purpose of this section is to describe how message forwarding interacts with the SOAP distributed processing model.

SOAP defines two different types of intermediaries: forwarding intermediaries and active intermediaries. These two types of intermediary are described in this section.

2.7.1 Relaying SOAP Header Blocks

The relaying of SOAP header blocks targeted at an intermediary SOAP node depends on whether the SOAP header blocks are processed or not by that node. A SOAP header block is said to be reinserted if the processing of that header block determines that the header block is to be reinserted in the forwarded message. The specification for a SOAP header block may call for the header block to be relayed in the forwarded message if the header block is targeted at a role played by the SOAP intermediary, but not otherwise processed by the intermediary. Such header blocks are said to be relayable.

A SOAP header block MAY carry a `relay` *attribute information item* (see **5.2.4 SOAP relay Attribute**). When the value of such an *attribute information item* is "true", the header block is said to be relayable. The forwarding of relayable header blocks is described in section **2.7.2 SOAP Forwarding Intermediaries**.

The `relay` *attribute information item* has no effect on SOAP header blocks targeted at a role other than one assumed by a SOAP intermediary.

The `relay` *attribute information item* has no effect on the SOAP processing model when the header block also carries a `mustUnderstand` *attribute information item* with a value of "true".

The `relay` *attribute information item* has no effect on the processing of SOAP messages by the SOAP ultimate receiver.

Table 3 summarizes the forwarding behavior of a SOAP node for a given header block. Each row contains a different combination of the value of the header block's role attribute information item, whether the SOAP node is acting in that role and whether the header block has been understood and processed, and shows whether the header block will be forwarded or removed.

Table 3: SOAP Nodes Forwarding behavior

Role		Header block	
Short-name	**Assumed**	**Understood & Processed**	**Forwarded**
`next`	Yes	Yes	No, unless reinserted
		No	No, unless `relay` ="true"
user-defined	Yes	Yes	No, unless reinserted
		No	No, unless `relay` ="true"
	No	n/a	Yes
`ultimateReceiver`	Yes	Yes	n/a
		No	n/a
`none`	No	n/a	Yes

2.7.2 SOAP Forwarding Intermediaries

The semantics of one or more SOAP header blocks in a SOAP message, or the SOAP MEP used, MAY require that the SOAP message be forwarded to another SOAP node on behalf of the initiator of the inbound SOAP message. In this case, the processing SOAP node acts in the role of a SOAP forwarding intermediary.

Forwarding SOAP intermediaries MUST process the message according to the SOAP processing model defined in **2.6 Processing SOAP Messages**. In addition, when generating a SOAP message for the purpose of forwarding, they MUST:

1. Remove all processed SOAP header blocks.
2. Remove all non-relayable SOAP header blocks that were targeted at the forwarding node but ignored during processing.
3. Retain all relayable SOAP header blocks that were targeted at the forwarding node but ignored during processing.

Forwarding SOAP intermediaries MUST also obey the specification for the SOAP forwarding features being used. The specification for each such a feature MUST describe the required semantics, including the rules describing how the forwarded message is constructed. Such rules MAY describe placement of inserted or reinserted SOAP header blocks. Inserted SOAP header blocks might be indistinguishable from one or more of the header blocks removed by the intermediary. Processing is defined here in terms of re-inserting header blocks (rather than leaving them in place) to emphasize the need to process them at each SOAP

node along the SOAP message path.

2.7.2.1 Relayed Infoset

This section describes the behavior of SOAP forwarding intermediaries with respect to preservation of the XML infoset properties of a relayed SOAP message.

Unless overridden by the processing of SOAP features at an intermediary (see **2.7.2 SOAP Forwarding Intermediaries**), the following rules apply:

1. All XML infoset properties of a message MUST be preserved, except as specified in rules 2 through 22.
2. The *element information item* for a header block targeted at an intermediary MAY be removed, by that intermediary, from the [children] property of the SOAP `Header` *element information item* as detailed in **2.7.2 SOAP Forwarding Intermediaries**.
3. *Element information items* for additional header blocks MAY be added to the [children] property of the SOAP `Header` *element information item* as detailed in **2.7.2 SOAP Forwarding Intermediaries**.

   In this case, a SOAP `Header` *element information item* MAY be added, as the first member of the [children] property of the SOAP `Envelope` *element information item*, if it is NOT already present.
4. *White space character information items* MAY be removed from the [children] property of the SOAP `Envelope` *element information item*.
5. *White space character information items* MAY be added to the [children] property of the SOAP `Envelope` *element information item*.
6. *White space character information items* MAY be removed from the [children] property of the SOAP `Header` *element information item*.
7. *White space character information items* MAY be added to the [children] property of the SOAP `Header` *element information item*.
8. *Comment information items* MAY be added to the [children] property of the SOAP `Envelope` *element information item*.
9. *Comment information items* MAY be removed from the [children] property of the SOAP `Envelope` *element information item*.
10. *Comment information items* MAY be added to the [children] property of the SOAP `Header` *element information item*.
11. *Comment information items* MAY be removed from the [children] property of the SOAP `Header` *element information item*.
12. *Attribute information items* MAY be added to the [attributes] property of the SOAP `Envelope` *element information item*.
13. *Attribute information items* MAY be added to the [attributes] property of the

SOAP `Header` *element information item*.

14. *Attribute information items* MAY be added to the [namespace attributes] property of the SOAP `Envelope` *element information item*.

15. *Attribute information items* MAY be added to the [namespace attributes] property of the SOAP `Header` *element information item*.

16. SOAP `role` *attribute information items* that are present in the [attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.2 SOAP role Attribute**.

17. SOAP `mustUnderstand` *attribute information items* that are present in the [attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.3 SOAP mustUnderstand Attribute**.

18. SOAP `relay` *attribute information items* that are present in the [attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.4 SOAP relay Attribute**.

19. The [base URI] property of the *document information item* need not be maintained.

20. The [base URI] property of *element information items* MAY be changed or removed.

21. The [character encoding scheme] property of the *document information item* MAY be changed or removed.

22. All *namespace information items* in the [in-scope namespaces] of *element information items* MUST be preserved. Additional *namespace information items* MAY be added.

Note:

The rules above allow for signing of SOAP header blocks, the SOAP body, and combinations of SOAP header blocks and the SOAP body.

In the absence of a canonicalization algorithm to normalize the infoset transformations and if the "http://www.w3.org/TR/2001/REC-xml-c14n-20010315" canonicalization algorithm is used then items 1-6 and 11-14 are incompatible with signing the SOAP envelope and items 1, 2, 5, 6, 12 and 14 are incompatible with signing the SOAP header.

Similarly, if the "http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" canonicalization algorithm is used then items 7 and 8 are incompatible with signing the SOAP envelope and items 9 and 10 are incompatible with signing the SOAP header.

Note:

White space character information items are those whose [character code] property has a value of #x20, #x9, #xD or #xA.

2.7.3 SOAP Active Intermediaries

In addition to the processing performed by forwarding SOAP intermediaries, active SOAP intermediaries undertake additional processing that can modify the outbound SOAP message in ways *not* described in the inbound SOAP message. That is, they can undertake processing not described by SOAP header blocks in the incoming SOAP message. The potential set of services provided by an active SOAP intermediary includes, but is not limited to: security services, annotation services, and content manipulation services.

The results of such active processing could impact the interpretation of SOAP messages by downstream SOAP nodes. For example, as part of generating an outbound SOAP message, an active SOAP intermediary might have removed and encrypted some or all of the SOAP header blocks found in the inbound SOAP message. It is strongly recommended that SOAP features provided by active SOAP intermediaries be described in a manner that allows such modifications to be detected by affected SOAP nodes in the message path.

2.8 SOAP Versioning Model

The version of a SOAP message is identified by the XML expanded name of the child *element information item* of the *document information item*. A SOAP Version 1.2 message has a child *element information item* of the *document information item* with a [local name] of `Envelope` and a [namespace name] of "http://www.w3.org/2003/05/soap-envelope" (see **5.1 SOAP Envelope**).

A SOAP node determines whether it supports the version of a SOAP message on a per message basis. In this context "support" means understanding the semantics of that version of a SOAP envelope. The versioning model is directed only at the SOAP `Envelope` *element information item*. It does not address versioning of SOAP header blocks, encodings, protocol bindings, or anything else.

A SOAP node MAY support multiple envelope versions. However, when processing a message, a SOAP node MUST use the semantics defined by the version of that message.

If a SOAP node receives a message whose version is not supported it MUST generate a fault (see **5.4 SOAP Fault**) with a `Value` of `Code` set to "env:VersionMismatch". Any other malformation of the message construct MUST result in the generation of a fault with a `Value` of `Code` set to "env:Sender".

Appendix **A. Version Transition From SOAP/1.1 to SOAP Version 1.2** defines a mechanism for transitioning from SOAP/1.1 to SOAP Version 1.2 using the `Upgrade` *element information item* (see **5.4.7 VersionMismatch Faults**).

3. SOAP Extensibility Model

SOAP provides a simple messaging framework whose core functionality is concerned with providing extensibility. The extensibility mechanisms described below can be used to add capabilities found in richer messaging environments.

3.1 SOAP Features

A SOAP feature is an extension of the SOAP messaging framework. Although SOAP poses no constraints on the potential scope of such features, example features may include "reliability", "security", "correlation", "routing", and message exchange

patterns (MEPs) such as request/response, one-way, and peer-to-peer conversations.

The SOAP extensibility model provides two mechanisms through which features can be expressed: the SOAP Processing Model and the SOAP Protocol Binding Framework (see **2. SOAP Processing Model** and **4. SOAP Protocol Binding Framework**). The former describes the behavior of a single SOAP node with respect to the processing of an individual message. The latter mediates the act of sending and receiving SOAP messages by a SOAP node via an underlying protocol.

The SOAP Processing Model enables SOAP nodes that include the mechanisms necessary to implement one or more features to express such features within the SOAP envelope as SOAP header blocks (see **2.4 Understanding SOAP Header Blocks**). Such header blocks can be intended for any SOAP node or nodes along a SOAP message path (see **2.3 Targeting SOAP Header Blocks**). The combined syntax and semantics of SOAP header blocks are known as a SOAP module, and are specified according to the rules in **3.3 SOAP Modules**.

In contrast, a SOAP protocol binding operates between two adjacent SOAP nodes along a SOAP message path. There is no requirement that the same underlying protocol is used for all hops along a SOAP message path. In some cases, underlying protocols are equipped, either directly or through extension, with mechanisms for providing certain features. The SOAP Protocol Binding Framework provides a scheme for describing these features and how they relate to SOAP nodes through a binding specification (see **4. SOAP Protocol Binding Framework**).

Certain features might require end-to-end as opposed to hop-by-hop processing semantics. Although the SOAP Protocol Binding Framework allows end-to-end features to be expressed outside the SOAP envelope, no standard mechanism is provided for the processing by intermediaries of the resulting messages. A binding specification that expresses such features external to the SOAP envelope needs to define its own processing rules for those externally expressed features. A SOAP node is expected to conform to these processing rules (for example, describing what information is passed along with the SOAP message as it leaves the intermediary). The processing of SOAP envelopes in accordance with the SOAP Processing Model (see **2. SOAP Processing Model**) MUST NOT be overridden by binding specifications.

It is recommended that, where practical, end-to-end features be expressed as SOAP header blocks, so that the rules defined by the SOAP Processing Model can be employed.

3.1.1 Requirements on Features

The specification of a feature MUST include the following:

1. A URI used to name the feature. This enables the feature to be unambiguously referenced in description languages or during negotiation.

2. The information (state) required at each node to implement the feature.

3. The processing required at each node in order to fulfill the obligations of the feature including any handling of communication failures that might occur in the underlying protocol (see also **4.2 Binding Framework**).

4. The information to be transmitted from node to node.

See **3.2 SOAP Message Exchange Patterns (MEPs)** for additional requirements on MEP features.

3.2 SOAP Message Exchange Patterns (MEPs)

A Message Exchange Pattern (MEP) is a template that establishes a pattern for the exchange of messages between SOAP nodes. MEPs are a type of feature, and unless otherwise stated, references in this specification to the term "feature" apply also to MEPs. The request-response MEP specified in SOAP 1.2 Part 2 [SOAP Part 2] illustrates the specification of a MEP feature.

The specification of a message exchange pattern MUST:

- As mandated by **3.1.1 Requirements on Features**, provide a URI to name the MEP.
- Describe the life cycle of a message exchange conforming to the pattern.
- Describe the temporal/causal relationships, if any, of multiple messages exchanged in conformance with the pattern (e.g., responses follow requests and are sent to the originator of the request.)
- Describe the normal and abnormal termination of a message exchange conforming to the pattern.

Underlying protocol binding specifications can declare their support for one or more named MEPs.

MEPs are SOAP features, so an MEP specification MUST conform to the requirements for SOAP feature specifications (see **3.1.1 Requirements on Features**). An MEP specification MUST also include:

1. Any requirements to generate additional messages (such as responses to requests in a request/response MEP).
2. Rules for the delivery or other disposition of SOAP faults generated during the operation of the MEP.

3.3 SOAP Modules

The term "SOAP module" refers to the specification of the syntax and semantics of one or more SOAP header blocks. A SOAP module realizes zero or more SOAP features. A module specification adheres to the following rules. It:

1. MUST identify itself with a URI. This enables the module to be unambiguously referenced in description languages or during negotiation.
2. MUST declare the features provided by a module (see **3.1 SOAP Features**).
3. MUST clearly and completely specify the content and semantics of the SOAP header blocks used to implement the behavior in question, including if appropriate any modifications to the SOAP processing model. The SOAP extensibility model does not limit the extent to which SOAP can be extended.

Nor does it prevent extensions from modifying the SOAP processing model from that described in **2. SOAP Processing Model**

4. MAY utilize the property conventions defined in SOAP 1.2 Part 2 [SOAP Part 2], section A Convention for Describing Features and Bindings, in describing the functionality that the module provides. If these conventions are followed, the module specification MUST clearly describe the relationship between the abstract properties and their representations in the SOAP envelope. Note that it is possible to write a feature specification purely in terms of abstract properties, and then write a separate module specification which implements that feature, mapping the properties defined in the feature specification to SOAP header blocks in the SOAP module.

5. MUST clearly specify any known interactions with or changes to the interpretation of the SOAP body. Furthermore, it MUST clearly specify any known interactions with or changes to the interpretation of other SOAP features and SOAP modules. For example, we can imagine a module which encrypts and removes the SOAP body, inserting instead a SOAP header block containing a checksum and an indication of the encryption mechanism used. The specification for such a module would indicate that the decryption algorithm on the receiving side is to be run *prior* to any other modules which rely on the contents of the SOAP body.

4. SOAP Protocol Binding Framework

SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. The SOAP Protocol Binding Framework provides general rules for the specification of protocol bindings; the framework also describes the relationship between bindings and SOAP nodes that implement those bindings. The HTTP binding in SOAP 1.2 Part 2 [SOAP Part 2] illustrates the specification of a binding. Additional bindings can be created by specifications that conform to the binding framework introduced in this chapter.

A SOAP binding specification:

- Declares the features provided by a binding.
- Describes how the services of the underlying protocol are used to transmit SOAP message infosets.
- Describes how the services of the underlying protocol are used to honor the contract formed by the features supported by that binding.
- Describes the handling of all potential failures that can be anticipated within the binding.
- Defines the requirements for building a conformant implementation of the binding being specified.

A binding does not provide a separate processing model and does not constitute a SOAP node by itself. Rather a SOAP binding is an integral part of a SOAP node (see **2. SOAP Processing Model**).

4.1 Goals of the Binding Framework

The goals of the binding framework are:

1. To set out the requirements and concepts that are common to all binding specifications.

2. To facilitate homogeneous description in situations where multiple bindings support common features, promoting reuse across bindings.

3. To facilitate consistency in the specification of optional features.

Two or more bindings can offer a given optional feature, such as reliable delivery, using different means. One binding might exploit an underlying protocol that directly facilitates the feature (e.g., the protocol is reliable), and the other binding might provide the necessary logic itself (e.g., reliability is achieved via logging and retransmission). In such cases, the feature can be made available to applications in a consistent manner, regardless of which binding is used.

4.2 Binding Framework

The creation, transmission, and processing of a SOAP message, possibly through one or more intermediaries, is specified in terms of a distributed state machine. The state consists of information known to a SOAP node at a given point in time, including but not limited to the contents of messages being assembled for transmission or received for processing. The state at each node can be updated either by local processing, or by information received from an adjacent node.

Section **2. SOAP Processing Model** of this specification describes the processing that is common to all SOAP nodes when receiving a message. The purpose of a binding specification is to augment those core SOAP rules with additional processing that is particular to the binding, and to specify the manner in which the underlying protocol is used to transmit information between adjacent nodes in the message path.

The distributed state machine that manages the transmission of a given SOAP message through its message path is the combination of the core SOAP processing (see **2. SOAP Processing Model**) operating at each node, in conjunction with the binding specifications connecting each pair of nodes. A binding specification MUST enable one or more MEPs.

In cases where multiple features are supported by a binding specification, the specifications for those features MUST provide any information necessary for their successful use in combination. Similarly, any dependencies of one feature on another (i.e., if successful use of one feature depends on use or non-use of another) MUST be specified. This binding framework does not provide any explicit mechanism for controlling the use of such interdependent features.

The binding framework provides no fixed means of naming or typing the information comprising the state at a given node. Individual feature and binding specifications are free to adopt their own conventions for specifying state. Note, however, that consistency across bindings and features is likely to be enhanced in situations where multiple feature specifications adopt consistent conventions for representing state. For example, multiple features might benefit from a consistent specification for an authentication credential, a transaction ID, etc. The HTTP binding in SOAP 1.2 Part 2

[SOAP Part 2] illustrates one such convention.

As described in **5. SOAP Message Construct**, each SOAP message is specified as an XML infoset that consists of a *document information item* with exactly one child: the SOAP `Envelope` *element information item*. Therefore, the minimum responsibility of a binding in transmitting a message is to specify the means by which the SOAP message infoset is transferred to and reconstituted by the binding at the receiving SOAP node and to specify the manner in which the transmission of the envelope is effected using the facilities of the underlying protocol.

5. SOAP Message Construct provides that all SOAP envelopes are serializable using an XML 1.0 serialization, so XML 1.0 or later versions of XML MAY be used by bindings as the "on the wire" representation of the XML Infoset. However, the binding framework does not require that every binding use an XML serialization for transmission; compressed, encrypted, fragmented representations and so on can be used if appropriate. A binding, if using an XML serialization of the XML infoset, MAY mandate that a particular character encoding or set of encodings be used.

A binding, if using a XML serialization, must list the versions of XML used to serialize the infoset, or if it delegates this to other means (like media type description). To preserve interoperability, the list of supported XML versions should be exhaustive.

Bindings MAY provide for streaming when processing messages. That is, SOAP nodes MAY begin processing a received SOAP message as soon as the necessary information is available. SOAP processing is specified in terms of SOAP message infosets (see **5. SOAP Message Construct**). Although streaming SOAP receivers will acquire such XML infosets incrementally, SOAP processing MUST yield results identical to those that would have been achieved if the entire SOAP envelope were available prior to the start of processing. For example, as provided in **2.6 Processing SOAP Messages**, identification of targeted SOAP header blocks, and checking of all `mustUnderstand` attributes is to be done before successful processing can proceed. Depending on the representation used for the XML infoset, and the order in which it is transmitted, this rule might limit the degree to which streaming can be achieved.

Bindings MAY depend on state that is modeled as being outside of the SOAP message infoset (e.g., retry counts), and MAY transmit such information to adjacent nodes. For example, some bindings take a message delivery address (typically a URI) that is not within the envelope.

5. SOAP Message Construct

A SOAP message is specified as an XML infoset whose comment, element, attribute, namespace and character information items are able to be serialized as XML 1.0. Note, requiring that the specified information items in SOAP message infosets be serializable as XML 1.0 does NOT require that they be serialized using XML 1.0. A SOAP message Infoset consists of a *document information item* with exactly one member in its [children] property, which MUST be the SOAP `Envelope` *element information item* (see **5.1 SOAP Envelope**). This *element information item* is also the value of the [document element] property. The [notations] and [unparsed entities] properties are both empty. The Infoset Recommendation [XML InfoSet] allows for content not directly serializable using XML; for example, the character #x0 is not prohibited in the Infoset, but is disallowed in XML. The XML Infoset of a SOAP Message MUST correspond to an XML 1.0 serialization [XML 1.0].

The XML infoset of a SOAP message MUST NOT contain a *document type declaration information item*.

SOAP messages sent by initial SOAP senders MUST NOT contain *processing instruction information items*. SOAP intermediaries MUST NOT insert *processing instruction information items* in SOAP messages they relay. SOAP receivers receiving a SOAP message containing a *processing instruction information item* SHOULD generate a SOAP fault with the `Value` of `Code` set to "env:Sender". However, in the case where performance considerations make it impractical for an intermediary to detect *processing instruction information items* in a message to be relayed, the intermediary MAY leave such *processing instruction information items* unchanged in the relayed message.

Element information items defined by this specification that only have *element information items* defined as allowable members of their [children] property can also have zero or more *character information item* children. The character code of each such *character information item* MUST be amongst the white space characters as defined by XML 1.0 [XML 1.0]. Unless otherwise indicated, such *character information items* are considered insignificant.

Comment information items MAY appear as children and/or descendants of the [document element] *element information item* but not before or after that *element information item* . There are some restrictions in the processing model with respect to when *comment information items* can be added and/or removed (see **2.7.2.1 Relayed Infoset**).

5.1 SOAP Envelope

The SOAP `Envelope` *element information item* has:

- A [local name] of `Envelope` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace-qualified *attribute information items* amongst its [attributes] property.
- One or two *element information item*s in its [children] property in order as follows:
    1. An optional `Header` *element information item* (see **5.2 SOAP Header**).
    2. A mandatory `Body` *element information item* (see **5.3 SOAP Body**).

5.1.1 SOAP encodingStyle Attribute

The `encodingStyle` *attribute information item* indicates the encoding rules used to serialize parts of a SOAP message.

The `encodingStyle` *attribute information item* has:

- A [local name] of `encodingStyle` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".

The `encodingStyle` *attribute information item* is of type *xs:anyURI*. Its value identifies a set of serialization rules that can be used to deserialize the SOAP message.

The `encodingStyle` *attribute information item* MAY appear on the following:

1. A SOAP header block (see **5.2.1 SOAP header block**).
2. A child *element information item* of the SOAP `Body` *element information item* (see **5.3.1 SOAP Body child Element**) if that child is not a SOAP Fault *element information item* (see **5.4 SOAP Fault**).
3. A child *element information item* of the SOAP `Detail` *element information item* (see **5.4.5.1 SOAP detail entry**).
4. Any descendent of 1, 2, and 3 above.

The `encodingStyle` *attribute information item* MUST NOT appear on any element other than above in a SOAP message infoset.

The scope of the `encodingStyle` *attribute information item* is that of its [owner element] and that *element information item's* descendants, excluding the scope of any inner `encodingStyle` *attribute information item*. If no `encodingStyle` *attribute information item* is in scope for a particular *element information item* or the value of such an *attribute information item* is "http://www.w3.org/2003/05/soap-envelope /encoding/none" then no claims are made regarding the encoding style of that *element information item* and its descendants.

Example 2: Values for the `encodingStyle` *attribute information item*.

```
"http://www.w3.org/2003/05/soap-encoding"
"http://example.org/encoding/"
"http://www.w3.org/2003/05/soap-envelope/encoding/none"
```

5.2 SOAP Header

The SOAP `Header` *element information item* provides a mechanism for extending a SOAP message in a decentralized and modular way (see **3. SOAP Extensibility Model** and **2.4 Understanding SOAP Header Blocks**).

The `Header` *element information item* has:

- A [local name] of `Header` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace-qualified *attribute information items* in its [attributes] property.
- Zero or more namespace-qualified *element information item*s in its [children] property.

Each child *element information item* of the SOAP Header is called a SOAP header block.

5.2.1 SOAP header block

Each SOAP header block *element information item*:

- MUST have a [namespace name] property which has a value; that is, the name of the element MUST be namespace-qualified.
- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have any number of *element information item* children. Such *element information items* MAY be namespace-qualified.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be any or all of the following, which have special significance for SOAP processing:
  - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).
  - `role` *attribute information item* (see **5.2.2 SOAP role Attribute**).
  - `mustUnderstand` *attribute information item* (see **5.2.3 SOAP mustUnderstand Attribute**).
  - `relay` *attribute information item* (see **5.2.4 SOAP relay Attribute**).

Example 3: SOAP Header with a single SOAP header block

```
<env:Header xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <t:Transaction xmlns:t="http://example.org/2001/06/tx"
                env:mustUnderstand="true" >
   5
 </t:Transaction>
</env:Header>
```

5.2.2 SOAP role Attribute

A SOAP role is used to indicate the SOAP node to which a particular SOAP header block is targeted (see **2.2 SOAP Roles and SOAP Nodes**).

The `role` *attribute information item* has the following XML infoset properties:

- A [local name] of `role` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the `role` *attribute information item* is *xs:anyURI*. The value of the `role` *attribute information item* is a URI that names a role that a SOAP node can assume.

Omitting the SOAP `role` *attribute information item* is equivalent to supplying that attribute with a value of "http://www.w3.org/2003/05/soap-envelope /role/ultimateReceiver".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept, the SOAP `role` *attribute information item* with a value of "http://www.w3.org/2003/05

/soap-envelope/role/ultimateReceiver".

If relaying the message, a SOAP intermediary MAY omit a SOAP `role` *attribute information item* if its value is "http://www.w3.org/2003/05/soap-envelope /role/ultimateReceiver" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `role` *attribute information item* only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.2.3 SOAP mustUnderstand Attribute

The SOAP `mustUnderstand` *attribute information item* is used to indicate whether the processing of a SOAP header block is mandatory or optional (see **2.4 Understanding SOAP Header Blocks**)

The `mustUnderstand` *attribute information item* has the following XML infoset properties:

- A [local name] of `mustUnderstand` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the `mustUnderstand` *attribute information item* is *xs:boolean*.

Omitting this *attribute information item* is defined as being semantically equivalent to including it with a value of "false".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept, the SOAP `mustUnderstand` *attribute information item* with a value of "false" or "0".

If generating a SOAP `mustUnderstand` *attribute information item*, a SOAP sender SHOULD use the canonical representation "true" of the attribute value (see XML Schema [XML Schema Part 2]). A SOAP receiver MUST accept any valid lexical representation of the attribute value.

If relaying the message, a SOAP intermediary MAY substitute "true" for the value "1", or "false" for "0". In addition, a SOAP intermediary MAY omit a SOAP `mustUnderstand` *attribute information item* if its value is "false" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `mustUnderstand` *attribute information item* only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.2.4 SOAP relay Attribute

The SOAP `relay` *attribute information item* is used to indicate whether a SOAP header block targeted at a SOAP receiver must be relayed if not processed (see **2.7.1 Relaying SOAP Header Blocks**).

The `relay` *attribute information item* has the following XML infoset properties:

- A [local name] of `relay` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the relay *attribute information item* is *xs:boolean*.

Omitting this *attribute information item* is defined as being semantically equivalent to including it with a value of "false".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept, the SOAP `relay` *attribute information item* with a value of "false" or "0".

If generating a SOAP `relay` *attribute information item*, a SOAP sender SHOULD use the canonical representation "true" of the attribute value (see XML Schema [XML Schema Part 2]). A SOAP receiver MUST accept any valid lexical representation of the attribute value.

If relaying the message, a SOAP intermediary MAY substitute "true" for the value "1", or "false" for "0". In addition, a SOAP intermediary MAY omit a SOAP `relay` *attribute information item* if its value is "false" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `relay` *attribute information item* only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.3 SOAP Body

A SOAP body provides a mechanism for transmitting information to an ultimate SOAP receiver (see **2.5 Structure and Interpretation of SOAP Bodies**).

The `Body` *element information item* has:

- A [local name] of `Body` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace-qualified *attribute information items* in its [attributes] property.
- Zero or more namespace-qualified *element information item*s in its [children] property.

The `Body` *element information item* MAY have any number of *character information item* children. The character code of such *character information items* MUST be amongst the white space characters as defined by XML 1.0 [XML 1.0]. These are considered significant.

5.3.1 SOAP Body child Element

All child *element information items* of the SOAP `Body` *element information item*:

- SHOULD have a [namespace name] property which has a value; that is, the

name of the element SHOULD be namespace-qualified.

Note:

Namespace-qualified elements tend to produce messages whose interpretation is less ambiguous than those with unqualified elements. The use of unqualified elements is therefore discouraged.

- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have any number of *element information item* children. Such *element information items* MAY be namespace-qualified.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be the following, which has special significance for SOAP processing:
  - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).

SOAP defines one particular direct child of the SOAP body, the SOAP fault, which is used for reporting errors (see **5.4 SOAP Fault**).

5.4 SOAP Fault

A SOAP fault is used to carry error information within a SOAP message.

The `Fault` *element information item* has:

- A [local name] of `Fault` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Two or more child *element information item*s in its [children] property in order as follows:
  1. A mandatory `Code` *element information item* (see **5.4.1 SOAP Code Element**).
  2. A mandatory `Reason` *element information item* (see **5.4.2 SOAP Reason Element**).
  3. An optional `Node` *element information item* (see **5.4.3 SOAP Node Element**).
  4. An optional `Role` *element information item* (see **5.4.4 SOAP Role Element**).
  5. An optional `Detail` *element information item* (see **5.4.5 SOAP Detail Element**).

To be recognized as carrying SOAP error information, a SOAP message MUST contain a single SOAP `Fault` *element information item* as the only child *element*

information item of the SOAP `Body` .

When generating a fault, SOAP senders MUST NOT include additional *element information items* in the SOAP `Body` . A message whose `Body` contains a `Fault` plus additional *element information items* has no SOAP-defined semantics.

A SOAP `Fault` *element information item* MAY appear within a SOAP header block, or as a descendant of a child *element information item* of the SOAP `Body` ; in such cases, the element has no SOAP-defined semantics.

5.4.1 SOAP Code Element

The `Code` *element information item* has:

- A [local name] of `Code` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or two child *element information item*s in its [children] property, in order, as follows:
    1. A mandatory `Value` *element information item* as described below (see **5.4.1.1 SOAP Value element (with Code parent)**)
    2. An optional `Subcode` *element information item* as described below (see **5.4.1.2 SOAP Subcode element**).

5.4.1.1 SOAP Value element (with Code parent)

The `Value` *element information item* has:

- A [local name] of `Value` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Value` *element information item* is *env:faultCodeEnum*. SOAP defines a small set of SOAP fault codes covering high level SOAP faults (see **5.4.6 SOAP Fault Codes**).

5.4.1.2 SOAP Subcode element

The `Subcode` *element information item* has:

- A [local name] of `Subcode` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or two child *element information item*s in its [children] property, in order, as follows:
    1. A mandatory `Value` *element information item* as described below (see **5.4.1.3 SOAP Value element (with Subcode parent)**).
    2. An optional `Subcode` *element information item* (see **5.4.1.2 SOAP**

Subcode element).

5.4.1.3 SOAP Value element (with Subcode parent)

The `Value` *element information item* has:

- A [local name] of `Value` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Value` *element information item* is *xs:QName*. The value of this element is an application defined subcategory of the value of the `Value` child *element information item* of the `Subcode` *element information item's* parent *element information item* (see **5.4.6 SOAP Fault Codes**).

5.4.2 SOAP Reason Element

The `Reason` *element information item* is intended to provide a human-readable explanation of the fault.

The `Reason` *element information item* has:

- A [local name] of `Reason` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or more `Text` *element information item* children (see **5.4.2.1 SOAP Text Element**). Each child `Text` *element information item* SHOULD have a different value for its `xml:lang` *attribute information item*.

The type of the `Reason` *element information item* is *env:faultReason*.

5.4.2.1 SOAP Text Element

The `Text` *element information item* is intended to carry the text of a human-readable explanation of the fault.

The `Text` *element information item* has:

- A [local name] of `Text` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- A mandatory *attribute information item* with a [local name] of `lang` and [namespace name] of "http://www.w3.org/XML/1998/namespace". Note that the definition in of the `lang` *attribute information item* requires that the [prefix] is "xml" or any capitalization thereof (see XML 1.0 [XML 1.0], Language Identification).
- Any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.

The type of the `Text` *element information item* is *env:reasontext*

This *element information item* is similar to the 'Reason-Phrase' defined by HTTP [RFC 2616] and SHOULD provide information explaining the nature of the fault. It is not intended for algorithmic processing.

5.4.3 SOAP Node Element

The `Node` *element information item* is intended to provide information about which SOAP node on the SOAP message path caused the fault to happen (see **2. SOAP Processing Model**).

The `Node` *element information item* has:

- A [local name] of `Node` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Node` *element information item* is *xs:anyURI*.

As described in section **2.1 SOAP Nodes**, each SOAP node is identified by a URI. The value of the `Node` *element information item* is the URI that identifies the SOAP node that generated the fault. SOAP nodes that do not act as the ultimate SOAP receiver MUST include this *element information item*. An ultimate SOAP receiver MAY include this *element information item* to indicate explicitly that it generated the fault.

5.4.4 SOAP Role Element

The `Role` *element information item* identifies the role the node was operating in at the point the fault occurred.

The `Role` *element information item* has:

- A [local name] of `Role` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Role` *element information item* is *xs:anyURI*.

The value of the `Role` *element information item* MUST be one of the roles assumed by the node during processing of the message (see **2.2 SOAP Roles and SOAP Nodes**).

5.4.5 SOAP Detail Element

The `Detail` *element information item* is intended for carrying application specific error information.

The `Detail` *element information item* has:

- A [local name] of `Detail` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- Zero or more *attribute information item*s in its [attributes] property.
- Zero or more child *element information item*s in its [children] property.

The `Detail` *element information item* MAY have any number of *character information item* children. The character code of each such *character information item* MUST be amongst the white space characters as defined by XML 1.0 [XML 1.0]. These are considered significant.

The `Detail` *element information item* MAY be present in a SOAP fault in which case it carries additional information relative to the SOAP fault codes describing the fault (see **5.4.6 SOAP Fault Codes**). For example, the `Detail` *element information item* might contain information about a message not containing the proper credentials, a timeout, etc. The presence of the `Detail` *element information item* has no significance as to which parts of the faulty SOAP message were processed.

All child *element information items* of the `Detail` *element information item* are called detail entries (see **5.4.5.1 SOAP detail entry**).

5.4.5.1 SOAP detail entry

Each detail entry:

- MAY have a [namespace name] property which has a value; that is, the name of the element MAY be namespace-qualified.
- MAY have any number of *element information item* children.
- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be the following, which has special significance for SOAP processing:
    - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).

If present, the SOAP `encodingStyle` *attribute information item* indicates the encoding style used for the detail entry (see **5.1.1 SOAP encodingStyle Attribute**).

5.4.6 SOAP Fault Codes

SOAP fault codes are XML expanded names, and are intended to provide a means by which faults are classified. A hierarchical list of SOAP codes and associated supporting information is included in every SOAP fault message, with each such code identifying the fault category at an increasing level of detail.

The values of the `Value` child *element information item* of the `Code` *element information item* are restricted to those defined by the `env:faultCodeEnum` type (see **Table 4**). Additional fault subcodes MAY be created for use by applications or features. Such subcodes are carried in the `Value` child *element information item* of the `Subcode` *element information item*.

SOAP fault codes are to be interpreted as modifiers of the contents of the `Detail` *element information item* in the sense that they provide the context for the `Detail` *element information item*. A SOAP node MUST understand all SOAP fault codes in a

SOAP fault message in order to be able to interpret the `Detail` *element information item* in a SOAP fault.

Example 4: Sample SOAP fault where the `Detail` *element information item* is to be interpreted in the context of the "env:Sender" and "m:MessageTimeout" fault codes.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:m="http://www.example.org/timeouts"
              xmlns:xml="http://www.w3.org/XML/1998/namespace">
 <env:Body>
  <env:Fault>
   <env:Code>
     <env:Value>env:Sender</env:Value>
     <env:Subcode>
      <env:Value>m:MessageTimeout</env:Value>
     </env:Subcode>
   </env:Code>
   <env:Reason>
     <env:Text xml:lang="en">Sender Timeout</env:Text>
   </env:Reason>
   <env:Detail>
     <m:MaxTime>P5M</m:MaxTime>
   </env:Detail>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

This specification does not define a limit for how many `Subcode` *element information items* a SOAP fault might contain. However, while not a requirement of this specification, it is anticipated that most practical examples can be supported by relatively few `Subcode` *element information items*.

Table 4: SOAP Fault Codes

Local Name	Meaning
VersionMismatch	The faulting node found an invalid *element information item* instead of the expected `Envelope` *element information item*. The namespace, local name or both did not match the `Envelope` *element information item* required by this recommendation (see **2.8 SOAP Versioning Model** and **5.4.7 VersionMismatch Faults**)
MustUnderstand	An immediate child *element information item* of the SOAP `Header` *element information item* targeted at the faulting node that was not understood by the faulting node contained a SOAP `mustUnderstand` *attribute information item* with a value of "true" (see **5.2.3 SOAP mustUnderstand Attribute** and **5.4.8 SOAP mustUnderstand Faults**)
DataEncodingUnknown	A SOAP header block or SOAP body child *element information item* targeted at the faulting SOAP node is scoped (see **5.1.1 SOAP encodingStyle Attribute**) with a data encoding that the faulting node does not support.
Sender	The message was incorrectly formed or did not contain the appropriate information in order to succeed. For example, the message could lack the proper authentication or payment information. It is generally an indication that the message is not to be resent without change (see also **5.4 SOAP Fault** for a description of the SOAP fault `detail` sub-element).
Receiver	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. For example, processing could include communicating with an upstream SOAP node, which did not respond. The message could succeed if resent at a later point in time (see also **5.4 SOAP Fault** for a description of the SOAP fault `detail` sub-element).

5.4.7 VersionMismatch Faults

When a SOAP node generates a fault with a `Value` of `Code` set to "env:VersionMismatch", it SHOULD provide an `Upgrade` SOAP header block in the generated fault message. The `Upgrade` SOAP header block, as described below, details the XML qualified names (per XML Schema [XML Schema Part 2]) of the supported SOAP envelopes that the SOAP node supports (see **2.8 SOAP Versioning Model**).

5.4.7.1 SOAP Upgrade Header Block

The `Upgrade` SOAP header block consists of an `Upgrade` *element information item* containing an ordered list of XML qualified names of SOAP envelopes that the SOAP node supports in the order most to least preferred.

The `Upgrade` *element information item* has:

- A [local name] of `Upgrade` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- One or more `SupportedEnvelope` *element information item*s in its [children] property in **5.4.7.2 SOAP SupportedEnvelope Element**.

The `Upgrade` *element information item* MUST NOT have an `encodingStyle` *attribute information item*.

5.4.7.2 SOAP SupportedEnvelope Element

The `SupportedEnvelope` *element information item* has:

- A [local name] of `SupportedEnvelope` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A `qname` *attribute information item* in its [attributes] property as described in **5.4.7.3 SOAP QName Attribute**.

5.4.7.3 SOAP QName Attribute

The `qname` *attribute information item* has the following XML infoset properties:

- A [local name] of `qname` .
- A [namespace name] which has no value.
- A [specified] property with a value of "true".

The type of the `qname` *attribute information item* is *xs:QName*. Its value is the XML qualified name of a SOAP `Envelope` *element information item* that the faulting node can understand.

Note:

When serializing the `qname` *attribute information item* there needs to be an in-scope namespace declaration for the namespace name of the SOAP `Envelope` *element information item* that the faulting node can understand. The value of the *attribute information item* uses the prefix of such a namespace declaration.

5.4.7.4 VersionMismatch Example

The following example illustrates the case of a SOAP node that supports both SOAP Version 1.2 and SOAP/1.1 but which prefers SOAP Version 1.2 (see appendix **A. Version Transition From SOAP/1.1 to SOAP Version 1.2** for a mechanism for transitioning from SOAP/1.1 to SOAP Version 1.2). This is indicated by including an `Upgrade` SOAP header block with two `SupportedEnvelope` *element information items*, the first containing the local name and namespace name of the SOAP Version 1.2 `Envelope` *element information item*, the latter the local name and namespace name of the SOAP/1.1 `Envelope` element.

Example 5: Version mismatch fault generated by a SOAP node. The message includes a SOAP `Upgrade` header block indicating support for both SOAP Version 1.2 and SOAP/1.1 but with a preference for SOAP Version 1.2.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:xml="http://www.w3.org/XML/1998/namespace">
 <env:Header>
  <env:Upgrade>
   <env:SupportedEnvelope qname="ns1:Envelope"
             xmlns:ns1="http://www.w3.org/2003/05/soap-envelope"/>
   <env:SupportedEnvelope qname="ns2:Envelope"
             xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"/>
  </env:Upgrade>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code><env:Value>env:VersionMismatch</env:Value></env:Code>
    <env:Reason>
      <env:Text xml:lang="en">Version Mismatch</env:Text>
    </env:Reason>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

5.4.8 SOAP mustUnderstand Faults

When a SOAP node generates a fault with a `Value` of `Code` set to "env:MustUnderstand", it SHOULD provide `NotUnderstood` SOAP header blocks in the generated fault message. The `NotUnderstood` SOAP header blocks, as described below, detail the XML qualified names (per XML Schema [XML Schema Part 2]) of the particular SOAP header block(s) which were not understood.

A SOAP node MAY generate a SOAP fault for any one or more SOAP header blocks that were not understood in a SOAP message. It is not a requirement that the fault contain the XML qualified names of all such SOAP header blocks.

5.4.8.1 SOAP NotUnderstood Element

Each `NotUnderstood` header block *element information item* has:

- A [local name] of `NotUnderstood` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A `qname` *attribute information item* in its [attributes] property as described in **5.4.8.2 SOAP QName Attribute**.

The `NotUnderstood` *element information item* MUST NOT have an `encodingStyle` *attribute information item*.

5.4.8.2 SOAP QName Attribute

The `qname` *attribute information item* has the following XML infoset properties:

- A [local name] of `qname` .
- A [namespace name] which has no value.
- A [specified] property with a value of "true".

The type of the `qname` *attribute information item* is *xs:QName*. Its value is the XML qualified name of a SOAP header block which the faulting node failed to understand.

Note:

When serializing the `qname` *attribute information item* there needs to be an in-scope namespace declaration for the namespace name of the SOAP header block that was not understood and the value of the *attribute information item* uses the prefix of such a namespace declaration. The prefix used need not be the same as the one used in the SOAP message that was not understood.

5.4.8.3 NotUnderstood Example

Consider the following example message:

Example 6: SOAP envelope that will cause a fault if `Extension1` or `Extension2` are not understood

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2003/05/soap-envelope'>
  <env:Header>
    <abc:Extension1 xmlns:abc='http://example.org/2001/06/ext'
       env:mustUnderstand='true'/>
    <def:Extension2 xmlns:def='http://example.com/stuff'
       env:mustUnderstand='true' />
  </env:Header>
  <env:Body>
    . . .
  </env:Body>
</env:Envelope>
```

The message in the above example will result in the fault message shown in the example below if the ultimate receiver of the SOAP message does not understand the two SOAP header blocks `abc:Extension1` and `def:Extension2` .

Example 7: SOAP fault generated as a result of not understanding `Extension1` and `Extension2`

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2003/05/soap-envelope'
              xmlns:xml='http://www.w3.org/XML/1998/namespace'>
 <env:Header>
  <env:NotUnderstood qname='abc:Extension1'
                     xmlns:abc='http://example.org/2001/06/ext' />
  <env:NotUnderstood qname='def:Extension2'
                     xmlns:def='http://example.com/stuff' />
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code><env:Value>env:MustUnderstand</env:Value></env:Code>
   <env:Reason>
     <env:Text xml:lang='en'>One or more mandatory
       SOAP header blocks not understood
     </env:Text>
   </env:Reason>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

6. Use of URIs in SOAP

SOAP uses URIs for some identifiers including, but not limited to, values of the `encodingStyle` (see **5.1.1 SOAP encodingStyle Attribute**) and `role` (see **5.2.2 SOAP role Attribute**) *attribute information items*. To SOAP, a URI is simply a formatted string that identifies a web resource.

Where this specification calls for a URI, the string supplied MUST conform to the URI syntax as described by RFC 3986 [RFC 3986]. Note: RFC 3987 [RFC 3987] provides means by which Internationalized Resource Identifiers, IRIs, can be encoded into corresponding URIs.

Although this section only applies to URIs directly used by *information items* defined by this specification, it is RECOMMENDED that application-defined data carried within a SOAP envelope use the same mechanisms and guidelines defined here for handling URIs.

URIs used as values in *information items* identified by the "http://www.w3.org/2003/05/soap-envelope" and "http://www.w3.org/2003/05/soap-encoding" XML namespaces can be either relative or absolute.

SOAP does not define a base URI but relies on the mechanisms defined in XML Base [XML Base] and RFC 3986 [RFC 3986] for establishing a base URI against which relative URIs can be made absolute.

The underlying protocol binding MAY define a base URI which can act as the base URI for the SOAP envelope (see **4. SOAP Protocol Binding Framework** and SOAP 1.2 Part 2 [SOAP Part 2], section HTTP binding).

SOAP does not define any equivalence rules for URIs in general as these are defined by the individual URI schemes and by RFC 3986 [RFC 3986]. However, because of

inconsistencies with respect to URI equivalence rules in many current URI parsers, it is RECOMMENDED that SOAP senders do not rely on any special equivalence rules in SOAP receivers in order to determine equivalence between URI values used in a SOAP message.

The use of IP addresses in URIs SHOULD be avoided whenever possible (see RFC 1900 [RFC 1900]). However, when used, the literal format for IPv6 addresses in URIs as described by RFC 3986 [RFC 3986] SHOULD be supported.

SOAP does not place any a priori limit on the length of a URI. Any SOAP node MUST be able to handle the length of any URI that it publishes and both SOAP senders and SOAP receivers SHOULD be able to deal with URIs of at least 2048 characters in length.

7. Security Considerations

The SOAP Messaging Framework does not directly provide any mechanisms for dealing with access control, confidentiality, integrity and non-repudiation. Such mechanisms can be provided as SOAP extensions using the SOAP extensibility model (see **3. SOAP Extensibility Model**). This section describes the security considerations that designers and implementors need to take into consideration when designing and using such mechanisms.

SOAP implementors need to anticipate rogue SOAP applications sending intentionally malicious data to a SOAP node (see **2. SOAP Processing Model**). It is strongly recommended that a SOAP node receiving a SOAP message is capable of evaluating to what level it can trust the sender of that SOAP message and its contents.

7.1 SOAP Nodes

SOAP can carry application-defined data as SOAP header blocks or as SOAP body contents. Processing a SOAP header block might include dealing with side effects such as state changes, logging of information, or the generation of additional messages. It is strongly recommended that, for any deployment scenario, only carefully specified SOAP header blocks with well understood security implications of any side effects be processed by a SOAP node.

Similarly, processing the SOAP body might imply the occurrence of side effects that could, if not properly understood, have severe consequences for the receiving SOAP node. It is strongly recommended that only well-defined body contents with known security implications be processed.

Security considerations, however, are not just limited to recognizing the immediate child elements of a SOAP header block and the SOAP body. Implementors need to pay special attention to the security implications of all data carried within a SOAP message that can cause the remote execution of any actions in the receiver's environment. This includes not only data expressed as XML infoset properties but data that might be encoded as property values including binary data or parameters, for example URI query strings. Before accepting data of any type, an application ought to be aware of the particular security implications associated with that data within the context it is being used.

SOAP implementors need to be careful to ensure that if processing of the various parts of a SOAP message is provided through modular software architecture, that

each module is aware of the overall security context. For example, the SOAP body ought not to be processed without knowing the context in which it was received.

7.2 SOAP Intermediaries

SOAP inherently provides a distributed processing model that might involve a SOAP message passing through multiple SOAP nodes (see **2. SOAP Processing Model**). SOAP intermediaries are by definition men in the middle, and represent an opportunity for man-in-the-middle attacks. Security breaches on systems that run SOAP intermediaries can result in serious security and privacy problems. A compromised SOAP intermediary, or an intermediary implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

In analyzing the security implications of potential SOAP-related security problems, it is important to realize that the scope of security mechanisms provided by the underlying protocol might not be the same scope as the whole message path of the SOAP message. There is no requirement in SOAP that all hops between participating SOAP nodes use the same underlying protocol and even if this were the case, the very use of SOAP intermediaries is likely to reach beyond the scope of transport-level security.

7.3 Underlying Protocol Bindings

The effects on security of not implementing a MUST or SHOULD, or doing something the specification says MUST NOT or SHOULD NOT be done can be very subtle. Binding specification authors ought to describe, in detail, the security implications of not following recommendations or requirements as most implementors will not have had the benefit of the experience and discussion that produced the specification (see **4. SOAP Protocol Binding Framework**).

In addition, a binding specification might not address or provide countermeasures for all aspects of the inherent security risks. The binding specification authors ought to identify any such risks as might remain and indicate where further countermeasures would be needed above and beyond those provided for in the binding specification.

Authors of binding specifications need to be aware that SOAP extension modules expressed as SOAP header blocks could affect the underlying protocol in unforeseen ways. A SOAP message carried over a particular protocol binding might result in seemingly conflicting features. An example of this is a SOAP message carried over HTTP, using the HTTP basic authentication mechanism in combination with a SOAP-based authentication mechanism. It is strongly recommended that a binding specification describes any such interactions between the extensions and the underlying protocols.

7.3.1 Binding to Application-Specific Protocols

Some underlying protocols could be designed for a particular purpose or application profile. SOAP bindings to such protocols MAY use the same endpoint identification (e.g., TCP port number) as the underlying protocol, in order to reuse the existing infrastructure associated with that protocol.

However, the use of well-known ports by SOAP might incur additional, unintended handling by intermediaries and underlying implementations. For example, HTTP is

commonly thought of as a "Web browsing" protocol, and network administrators might place certain restrictions upon its use, or could interpose services such as filtering, content modification, routing, etc. Often, these services are interposed using port number as a heuristic.

As a result, binding definitions for underlying protocols with well-known default ports or application profiles SHOULD document potential interactions with commonly deployed infrastructure at those default ports or in conformance with default application profiles. Binding definitions SHOULD also illustrate the use of the binding on a non-default port as a means of avoiding unintended interaction with such services.

8. References

8.1 Normative References

[SOAP Part 2]
SOAP Version 1.2 Part 2: Adjuncts (Second Edition), Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, Yves Lafon, Editors. World Wide Web Consortium, 27 April 2007. This version is http://www.w3.org/TR/2007/REC-soap12-part2-20070427. The latest version is available at http://www.w3.org/TR/soap12-part2/.

[RFC 2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk Nielsen, P. Leach, L. Masinter and T. Berners-Lee, Editors. IETF, June 1999. This RFC is available at http://www.ietf.org/rfc/rfc2616.txt.

[RFC 2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Editor. IETF, March 1997. This RFC is available at http://www.ietf.org/rfc/rfc2119.txt.

[XML Schema Part 1]
XML Schema Part 1: Structures Second Edition, David Beech, Murray Maloney, Henry S. Thompson, and Noah Mendelsohn, Editors. World Wide Web Consortium, 28 October 2004. This version is http://www.w3.org/TR/2004 /REC-xmlschema-1-20041028/ The latest version is available at http://www.w3.org/TR/xmlschema-1/.

[XML Schema Part 2]
XML Schema Part 2: Datatypes Second Edition, Ashok Malhotra and Paul V. Biron, Editors. World Wide Web Consortium, 28 October 2004. This version is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. The latest version is available at http://www.w3.org/TR/xmlschema-2/.

[RFC 3986]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding and L. Masinter, Editors. IETF, January 2005. *Obsoletes: RFC 2396, RFC 2732.* This RFC is available at http://www.ietf.org/rfc/rfc3986.txt.

[Namespaces in XML]
Namespaces in XML (Second Edition), Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin, Editors. World Wide Web Consortium, 16 August 2006. This version is http://www.w3.org/TR/2006/REC-xml-names-20060816. The latest version is available at http://www.w3.org/TR/REC-xml-names.

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Fourth Edition), Jean Paoli, Eve Maler, Tim Bray, *et. al.*, Editors. World Wide Web Consortium, 16 August 2006. This version is http://www.w3.org/TR/2006/REC-xml-20060816. The latest version is

available at http://www.w3.org/TR/REC-xml.

[XML InfoSet]
XML Information Set (Second Edition), Richard Tobin and John Cowan, Editors. World Wide Web Consortium, 04 February 2004. This version is http://www.w3.org/TR/2004/REC-xml-infoset-20040204. The latest version is available at http://www.w3.org/TR/xml-infoset.

[XML Base]
XML Base, Jonathan Marsh, Editor. World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version is available at http://www.w3.org/TR/xmlbase/.

8.2 Informative References

[SOAP Part 0]
SOAP Version 1.2 Part 0: Primer (Second Edition), Nilo Mitra, Yves Lafon, Editors. World Wide Web Consortium, 27 April 2007. This version is http://www.w3.org/TR/2007/REC-soap12-part0-20070427. The latest version is available at http://www.w3.org/TR/soap12-part0/.

[XMLP Comments]
XML Protocol Comments Archive (See http://lists.w3.org/Archives/Public/xmlp-comments/.)

[XMLP Dist-App]
XML Protocol Discussion Archive (See http://lists.w3.org/Archives/Public/xml-dist-app/.)

[XMLP Charter]
XML Protocol Charter (See http://www.w3.org/2005/07/XML-Protocol-Charter.)

[XMLP Requirements]
XML Protocol (XMLP) Requirements, Vidur Apparao, Alex Ceponkus, Paul Cotton, David Ezell, David Fallside, Martin Gudgin, Oisin Hurley, John Ibbotson, R. Alexander Milowski, Kevin Mitchell, Jean-Jacques Moreau, Eric Newcomer, Henrik Frystyk Nielsen, Mark Nottingham, Waqar Sadiq, Stuart Williams, Amr Yassin, Editors. World Wide Web Consortium, 28 July 2003. This version is http://www.w3.org/TR/2003/NOTE-xmlp-reqs-20030728/. The latest version is available at http://www.w3.org/TR/xmlp-reqs.

[SOAP Usage Scenarios]
SOAP Version 1.2 Usage Scenarios, John Ibbotson, Editor. World Wide Web Consortium, 30 July 2003. This version is http://www.w3.org/TR/2003/NOTE-xmlp-scenarios-20030730/. The latest version is available at http://www.w3.org/TR/xmlp-scenarios.

[SOAP 1.1]
Simple Object Access Protocol (SOAP) 1.1, Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Nielsen, Satish Thatte, Dave Winer, Editors. DevelopMentor, IBM, Microsoft, Lotus Development Corp., UserLand Software, Inc., 30 July 2003. This version is http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[RFC 3987]
Internationalized Resource Identifiers (IRIs) , M. Duerst, Editors. IETF, January 2005. This RFC is available at http://www.ietf.org/rfc/rfc3987.txt.

[RFC 1900]
Renumbering Needs Work, B. Carpenter, Y. Rekhter, Editors. Editor. IETF, February 1996. This RFC is available at http://www.ietf.org/rfc/rfc1900.txt.

A. Version Transition From SOAP/1.1 to SOAP Version 1.2

This appendix describes the version management rules for a SOAP node. If a SOAP node supports versioning from SOAP 1.1 to SOAP 1.2, then the SOAP node MUST implement the rules described in this appendix.

The rules for dealing with the possible SOAP/1.1 and SOAP Version 1.2 interactions are as follows:

1. A SOAP/1.1 node receiving a SOAP Version 1.2 message will according to SOAP/1.1 generate a version mismatch SOAP fault based on a SOAP/1.1 message construct. That is, the envelope will have a [local name] of `Envelope` and a [namespace name] of "http://schemas.xmlsoap.org/soap/envelope/".
2. A SOAP Version 1.2 node receiving a SOAP/1.1 message either:
    - MAY process the message as a SOAP/1.1 message (if supported), or
    - MUST generate a version mismatch SOAP fault based on a SOAP/1.1 message construct following SOAP/1.1 semantics using a SOAP/1.1 binding to the underlying protocol (see SOAP 1.1 [SOAP 1.1]). The SOAP fault SHOULD include an `Upgrade` SOAP header block as defined in this specification (see **5.4.7 VersionMismatch Faults**) indicating support for SOAP Version 1.2. This allows a receiving SOAP/1.1 node to correctly interpret the SOAP fault generated by the SOAP Version 1.2 node.

The example below shows a version mismatch SOAP fault generated by a SOAP Version 1.2 node as a result of receiving a SOAP/1.1 message. The fault message is a SOAP/1.1 message with an `Upgrade` SOAP header block indicating support for SOAP Version 1.2.

Example 8: SOAP Version 1.2 node generating a SOAP/1.1 version mismatch fault message including an `Upgrade` SOAP header block indicating support for SOAP Version 1.2.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header>
  <env:Upgrade>
   <env:SupportedEnvelope qname="ns1:Envelope"
             xmlns:ns1="http://www.w3.org/2003/05/soap-envelope"/>
  </env:Upgrade>
 </env:Header>
 <env:Body>
  <env:Fault>
   <faultcode>env:VersionMismatch</faultcode>
   <faultstring>Version Mismatch</faultstring>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

Note:

SOAP nodes wishing to support both SOAP/1.1 and SOAP Version 1.2 are required to use a protocol binding associated with the appropriate version of SOAP.

Note:

An existing SOAP/1.1 node generating a version mismatch SOAP fault is not likely to indicate which versions it supports using the `Upgrade` *element*

information item (see **5.4.7 VersionMismatch Faults**). If nothing is indicated then this means that SOAP/1.1 is the only supported version. Note, however that incompatibilities between underlying protocol bindings might prevent a SOAP/1.1 node from generating a version mismatch SOAP fault when receiving a SOAP Version 1.2 message. For instance, a SOAP/1.1 node supporting the SOAP/1.1 HTTP binding (see SOAP 1.1 [SOAP 1.1]) receiving a SOAP Version 1.2 message using the SOAP 1.2 HTTP protocol binding (see SOAP 1.2 Part 2 [SOAP Part 2], SOAP HTTP Binding) might not understand the difference between the two bindings and generate an HTTP specific response as a result.

B. Acknowledgements (Non-Normative)

This specification is the work of the W3C XML Protocol Working Group.

Participants in the Working Group are (at the time of writing, and by alphabetical order): Glen Daniels (Sonic Software, formerly of Macromedia), Vikas Deolaliker (Sonoa Systems, Inc.), Chris Ferris (IBM, formerly of Sun Microsystems), Marc Hadley (Sun Microsystems), David Hull (TIBCO Software, Inc.), Anish Karmarkar (Oracle), Yves Lafon (W3C), Jonathan Marsh (WSO2), Jeff Mischkinsky (Oracle), Eric Newcomer (IONA Technologies), David Orchard (BEA Systems, formerly of Jamcracker), Seumas Soltysik (IONA Technologies), Davanum Srinivas (WSO2), Pete Wenzel (Sun Microsystems, formerly of SeeBeyond).

Previous participants were: Yasser alSafadi (Philips Research), Bill Anderson (Xerox), Vidur Apparao (Netscape), Camilo Arbelaez (webMethods), Mark Baker (Idokorro Mobile, Inc., formerly of Sun Microsystems), Philippe Bedu (EDF (Electricite De France)), Olivier Boudeville (EDF (Electricite De France)), Carine Bournez (W3C), Don Box (Microsoft Corporation, formerly of DevelopMentor), Tom Breuel (Xerox), Dick Brooks (Group 8760), Winston Bumpus (Novell, Inc.), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Ceponkus (Bowstreet), Michael Champion (Software AG), David Chappell (Sonic Software), Miles Chaston (Epicentric), David Clay (Oracle), David Cleary (Progress Software), Dave Cleary (webMethods), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Fransisco Cubera (IBM), Jim d'Augustine (Excelon Corporation), Ron Daniel (Interwoven), Doug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE Corporation), Frank DeRose (TIBCO Software, Inc.), Mike Dierken (DataChannel), Andrew Eisenberg (Progress Software), Brian Eisenberg (DataChannel), Colleen Evans (Sonic Software), John Evdemon (XMLSolutions), David Ezell (Hewlett Packard), James Falek (TIBCO Software, Inc.), David Fallside (IBM), Eric Fedok (Active Data Exchange), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Michael Freeman (Engenia Software), Dietmar Gaertner (Software AG), Scott Golubock (Epicentric), Tony Graham (Sun Microsystems), Mike Greenberg (IONA Technologies), Rich Greenfield (Library of Congress), Martin Gudgin (Microsoft Corporation, formerly of DevelopMentor), Hugo Haas (W3C), Mark Hale (Interwoven), Randy Hall (Intel), Bjoern Heckel (Epicentric), Frederick Hirsch (Zolera Systems), Gerd Hoelzing (SAP AG), Erin Hoffmann (Tradia Inc.), Steve Hole (MessagingDirect Ltd.), Mary Holstege (Calico Commerce), Jim Hughes (Fujitsu Limited), Oisin Hurley (IONA Technologies), Yin-Leng Husband (Hewlett Packard, formerly of Compaq), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric Industrial Co., Ltd.), Scott Isaacson (Novell, Inc.), Kazunori Iwasa (Fujitsu Limited), Murali Janakiraman (Rogue Wave), Mario Jeckle (DaimlerChrysler Research and Technology), Eric Jenkins (Engenia Software), Mark Jones (AT&T), Jay Kasi (Commerce One), Jeffrey Kay (Engenia Software), Suresh Kodichath (IONA Technologies), Richard Koo (Vitria Technology Inc.), Jacek Kopecky (Systinet), Alan Kropp (Epicentric), Julian Kumar

(Epicentric), Peter Lecuyer (Progress Software), Tony Lee (Vitria Technology Inc.), Michah Lerner (AT&T), Bob Lojek (Intalio Inc.), Henry Lowe (OMG), Brad Lund (Intel), Matthew MacKenzie (XMLGlobal Technologies), Michael Mahan (Nokia), Murray Maloney (Commerce One), Richard Martin (Active Data Exchange), Noah Mendelsohn (IBM, formerly of Lotus Development), Alex Milowski (Lexica), Kevin Mitchell (XMLSolutions), Nilo Mitra (Ericsson), Ed Mooney (Sun Microsystems), Jean-Jacques Moreau (Canon), Dean Moses (Epicentric), Highland Mary Mountain (Intel), Don Mullen (TIBCO Software, Inc.), Rekha Nagarajan (Calico Commerce), Raj Nair (Cisco Systems), Masahiko Narita (Fujitsu Limited), Mark Needleman (Data Research Associates), Art Nevarez (Novell, Inc.), Henrik Nielsen (Microsoft Corporation), Mark Nottingham (BEA Systems, formerly of Akamai Technologies), Conleth O'Connell (Vignette), Kevin Perkins (Compaq), Doug Purdy (Microsoft Corporation), Jags Ramnaryan (BEA Systems), Andreas Riegg (DaimlerChrysler Research and Technology), Vilhelm Rosenqvist (NCR), Herve Ruellan (Canon), Marwan Sabbouh (MITRE Corporation), Waqar Sadiq (Vitria Technology Inc.), Rich Salz (Zolera Systems), Krishna Sankar (Cisco Systems), Jeff Schlimmer (Microsoft Corporation), George Scott (Tradia Inc.), Shane Sesta (Active Data Exchange), Lew Shannon (NCR), John-Paul Sicotte (MessagingDirect Ltd.), Miroslav Simek (Systinet), Simeon Simeonov (Macromedia), Aaron Skonnard (DevelopMentor), Nick Smilonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Anne Thomas Manes (Sun Microsystems), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Jim Trezzo (Oracle), Asir Vedamuthu (webMethods), Mike Vernal (Microsoft Corporation), Randy Waldrop (WebMethods), Fred Waskiewicz (OMG), David Webber (XMLGlobal Technologies), Ray Whitmer (Netscape), Volker Wiechers (SAP AG), Stuart Williams (Hewlett Packard), Yan Xu (DataChannel), Amr Yassin (Philips Research), Susan Yee (Active Data Exchange), Jin Yu (MartSoft Corp.).

The people who have contributed to discussions on xml-dist-app@w3.org are also gratefully acknowledged.