



SOAP Version 1.2 Part 2: Adjuncts

W3C Recommendation 24 June 2003

This version:

<http://www.w3.org/TR/2003/REC-soap12-part2-20030624/>

Latest version:

<http://www.w3.org/TR/soap12-part2/>

Previous versions:

<http://www.w3.org/TR/2003/PR-soap12-part2-20030507/>

Editors:

Martin Gudgin, Microsoft
Marc Hadley, Sun Microsystems
Noah Mendelsohn, IBM
Jean-Jacques Moreau, Canon
Henrik Frystyk Nielsen, Microsoft

Please refer to the [errata](#) for this document, which may include some normative corrections.

The English version of this specification is the only normative version. Non-normative [translations](#) may also be available.

[Copyright](#) ©2003 W3C[®](MIT, ERCIM, Keio), All Rights Reserved. W3C [viability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP Version 1.2 Part 2: Adjuncts defines a set of adjuncts that may be used with SOAP Version 1.2 Part 1: Messaging Framework. This specification depends on SOAP Version 1.2 Part 1: Messaging Framework [\[SOAP Part 1\]](#).

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is a [Recommendation](#) of the W3C. This document has been produced by the [XML Protocol Working Group](#), which is part of the [Web Services Activity](#). It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference 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.

Comments on this document are welcome. Please send them to the public mailing-list xmlp-comments@w3.org ([archive](#)). It is inappropriate to send discussion email to this address.

Information about implementations relevant to this specification can be found in the Implementation Report at <http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html>.

Patent disclosures relevant to this specification may be found on the Working Group's [patent disclosure page](#), in conformance with W3C 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 Data Model](#)
3. [SOAP Encoding](#)
4. [SOAP RPC Representation](#)
5. [A Convention for Describing Features and Bindings](#)
6. [SOAP-Supplied Message Exchange Patterns and Features](#)
7. [SOAP HTTP Binding](#)
8. [References](#)
 - A. [The application/soap+xml Media Type](#)
 - B. [Mapping Application Defined Names to XML Names](#)
 - C. [Using W3C XML Schema with SOAP Encoding](#) (Non-Normative)
 - D. [Acknowledgements](#) (Non-Normative)

Table of Contents

1. [Introduction](#)
 - 1.1 [Notational Conventions](#)
2. [SOAP Data Model](#)
 - 2.1 [Graph Edges](#)
 - 2.1.1 [Edge labels](#)
 - 2.2 [Graph Nodes](#)
 - 2.2.1 [Single and Multi Reference Nodes](#)
 - 2.3 [Values](#)
3. [SOAP Encoding](#)
 - 3.1 [Mapping between XML and the SOAP Data Model](#)
 - 3.1.1 [Encoding Graph Edges and Nodes](#)
 - 3.1.2 [Encoding Simple Values](#)
 - 3.1.3 [Encoding Compound Values](#)
 - 3.1.4 [Computing the Type Name Property](#)
 - 3.1.4.1 [itemType Attribute Information Item](#)
 - 3.1.5 [Unique identifiers](#)
 - 3.1.5.1 [id Attribute Information Item](#)
 - 3.1.5.2 [ref Attribute Information Item](#)
 - 3.1.5.3 [Constraints on id and ref Attribute Information Items](#)
 - 3.1.6 [arraySize Attribute Information Item](#)
 - 3.1.7 [nodeType Attribute Information Item](#)
 - 3.2 [Decoding Faults](#)
4. [SOAP RPC Representation](#)
 - 4.1 [Use of RPC on the World Wide Web](#)
 - 4.1.1 [Identification of RPC Resources](#)
 - 4.1.2 [Distinguishing Resource Retrievals from other RPCs](#)
 - 4.2 [RPC and SOAP Body](#)
 - 4.2.1 [RPC Invocation](#)
 - 4.2.2 [RPC Response](#)
 - 4.2.3 [SOAP Encoding Restriction](#)
 - 4.3 [RPC and SOAP Header](#)
 - 4.4 [RPC Faults](#)
5. [A Convention for Describing Features and Bindings](#)
 - 5.1 [Model and Properties](#)
 - 5.1.1 [Properties](#)
 - 5.1.2 [Property Scope](#)
 - 5.1.2.1 [Message Exchange Context](#)
 - 5.1.2.2 [Environment Context](#)
 - 5.1.3 [Properties and Features](#)
6. [SOAP-Supplied Message Exchange Patterns and Features](#)
 - 6.1 [Property Conventions for SOAP Message Exchange Patterns](#)
 - 6.2 [SOAP Request-Response Message Exchange Pattern](#)
 - 6.2.1 [SOAP Feature Name](#)
 - 6.2.2 [Description](#)
 - 6.2.3 [State Machine Description](#)
 - 6.2.4 [Fault Handling](#)
 - 6.3 [SOAP Response Message Exchange Pattern](#)

- 6.3.1 [SOAP Feature Name](#)
- 6.3.2 [Description](#)
- 6.3.3 [State Machine Description](#)
- 6.3.4 [Fault Handling](#)
- 6.4 [SOAP Web Method Feature](#)
 - 6.4.1 [SOAP Feature Name](#)
 - 6.4.2 [Description](#)
 - 6.4.3 [SOAP Web Method Feature State Machine](#)
- 6.5 [SOAP Action Feature](#)
 - 6.5.1 [SOAP Feature Name](#)
 - 6.5.2 [Description](#)
 - 6.5.3 [SOAP Action Feature State Machine](#)
- 7. [SOAP HTTP Binding](#)
 - 7.1 [Introduction](#)
 - 7.1.1 [Optionality](#)
 - 7.1.2 [Use of HTTP](#)
 - 7.1.3 [Interoperability with non-SOAP HTTP Implementations](#)
 - 7.1.4 [HTTP Media-Type](#)
 - 7.2 [Binding Name](#)
 - 7.3 [Supported Message Exchange Patterns](#)
 - 7.4 [Supported Features](#)
 - 7.5 [MEP Operation](#)
 - 7.5.1 [Behavior of Requesting SOAP Node](#)
 - 7.5.1.1 [Init](#)
 - 7.5.1.2 [Requesting](#)
 - 7.5.1.3 [Sending+Receiving](#)
 - 7.5.1.4 [Receiving](#)
 - 7.5.1.5 [Success and Fail](#)
 - 7.5.2 [Behavior of Responding SOAP Node](#)
 - 7.5.2.1 [Init](#)
 - 7.5.2.2 [Receiving](#)
 - 7.5.2.3 [Receiving+Sending](#)
 - 7.5.2.4 [Sending](#)
 - 7.5.2.5 [Success and Fail](#)
 - 7.6 [Security Considerations](#)
- 8. [References](#)
 - 8.1 [Normative References](#)
 - 8.2 [Informative References](#)

Appendices

- A. [The application/soap+xml Media Type](#)
 - A.1 [Registration](#)
 - A.2 [Security Considerations](#)
 - A.3 [The action Parameter](#)
- B. [Mapping Application Defined Names to XML Names](#)
 - B.1 [Rules for Mapping Application Defined Names to XML Names](#)
 - B.2 [Examples](#)

- C. [Using W3C XML Schema with SOAP Encoding](#) (Non-Normative)
 - C.1 [Validating Using the Minimum Schema](#)
 - C.2 [Validating Using the SOAP Encoding Schema](#)
 - C.3 [Validating Using More Specific Schemas](#)
 - D. [Acknowledgements](#) (Non-Normative)
-

1. Introduction

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchange of structured information in a decentralized, distributed environment. The SOAP specification consists of three parts. Part 2 (this document) defines a set of adjuncts that MAY be used with the SOAP messaging framework:

1. The SOAP Data Model represents application-defined data structures and values as a directed, edge-labeled graph of nodes (see [2. SOAP Data Model](#)).
2. The SOAP Encoding defines a set of rules for encoding instances of data that conform to the SOAP Data Model for inclusion in SOAP messages (see [3. SOAP Encoding](#)).
3. The SOAP RPC Representation defines a convention for how to use the SOAP Data Model for representing RPC calls and responses (see [4. SOAP RPC Representation](#)).
4. The section for describing features and bindings defines a convention for describing features and binding in terms of properties and property values (see [5. A Convention for Describing Features and Bindings](#)).
5. The section on SOAP-Supplied Message Exchange Patterns and Features defines a request response message exchange pattern and a message exchange pattern supporting non-SOAP requests for SOAP responses, (see [6. SOAP-Supplied Message Exchange Patterns and Features](#)).
6. The SOAP Web Method feature defines a feature for control of methods used on the World Wide Web (see [6.4 SOAP Web Method Feature](#)).
7. The SOAP HTTP Binding defines a binding of SOAP to HTTP (see [\[RFC 2616\]](#)) following the rules of the [SOAP Protocol Binding Framework, \[SOAP Part 1\]](#) (see [7. SOAP HTTP Binding](#)).

SOAP 1.2 Part 0 [\[SOAP Part 0\]](#) 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 1 [\[SOAP Part 1\]](#) defines 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"	Defined by SOAP 1.2 Part 1 [SOAP Part 1] .
enc	"http://www.w3.org/2003/05/soap-encoding"	A normative XML Schema [XML Schema Part 1] , [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-encoding" namespace can be found at http://www.w3.org/2003/05/soap-encoding .
rpc	"http://www.w3.org/2003/05/soap-rpc"	A normative XML Schema [XML Schema Part 1] , [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-rpc" namespace can be found at http://www.w3.org/2003/05/soap-rpc .
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [XML Schema Part 1] , [XML Schema Part 2] .
xsi	"http://www.w3.org/2001/XMLSchema-instance"	Defined in the W3C XML Schema specification [XML Schema Part 1] , [XML Schema Part 2] .

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs (see RFC 2396 [\[RFC 2396\]](#)).

This specification uses the Extended Backus-Naur Form (EBNF) as described in XML 1.0 [\[XML 1.0\]](#).

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

2. SOAP Data Model

The SOAP Data Model represents application-defined data structures and values as a directed edge-labeled graph of nodes. Components of this graph are described in the following sections.

The purpose of the SOAP Data Model is to provide a mapping of non-XML based data to some wire representation. It is important to note that use of the SOAP Data Model, the accompanying SOAP Encoding (see [3. SOAP Encoding](#)), and/or the SOAP RPC Representation (see [4. SOAP RPC Representation](#)) is OPTIONAL. Applications which already model data in XML may not need to use the SOAP Data Model. Due to their optional nature, it is NOT a requirement to implement the SOAP Data Model, the SOAP Encoding and/or the SOAP RPC Representation as part of a SOAP node.

2.1 Graph Edges

Edges in the graph are said to *originate* at a graph node and *terminate* at a graph node. An edge that originates at a graph node is known as an *outbound edge* with respect to that graph node. An edge that terminates at a graph node is known as an *inbound edge* with respect to that graph node. An edge MAY originate and terminate at the same graph node. An edge MAY have only an originating graph node, that is be outbound only. An edge MAY have only a terminating graph node, that is be inbound only.

The outbound edges of a given graph node MAY be distinguished by label or by position. Position is a total order on such edges; thus, if any outbound edges from a given node are distinguished by position, then all outbound edges from that node are so distinguished.

2.1.1 Edge labels

An edge label is an XML qualified name. Two edge labels are equal if and only if their XML expanded names are equal. I.e. both of the following are true:

1. Their local name values are the same.
2. Either of the following is true:
 1. Both of their namespace name values are missing.

2. Their namespace name values are both present and are both the same.

See [2.3 Values](#) for uses of edge labels and position to distinguish the members of encoded values, and XML Schema [\[XML Schema Part 2\]](#) for more information about comparing XML qualified names.

2.2 Graph Nodes

A graph node has zero or more outbound edges. A graph node that has no outbound edges has an optional lexical value. All graph nodes have an optional type name of type `xs:QName` in the namespace named "http://www.w3.org/2001/XMLSchema" (see XML Schema [\[XML Schema Part 2\]](#)).

2.2.1 Single and Multi Reference Nodes

A graph node may be *single reference* or *multi reference*. A single reference graph node has a single inbound edge. A multi reference graph node has multiple inbound edges.

2.3 Values

A simple value is represented as a graph node with a lexical value.

A compound value is represented as a graph node with zero or more outbound edges as follows:

1. A graph node whose outbound edges are distinguished solely by their labels is known as a "struct". The outbound edges of a struct MUST be labeled with distinct names (see [2.1.1 Edge labels](#)).
2. A graph node whose outbound edges are distinguished solely by position is known as an "array". The outbound edges of an array MUST NOT be labeled.

3. SOAP Encoding

SOAP Encoding provides a means of encoding instances of data that conform to the data model described in [2. SOAP Data Model](#). This encoding MAY be used to transmit data in SOAP header blocks and/or SOAP bodies. Other data models, alternate encodings of the SOAP Data Model as well as unencoded data MAY also be used in SOAP messages (see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), [SOAP encodingStyle Attribute](#) for specification of alternative encoding styles and see [4. SOAP RPC Representation](#) for restrictions on data models and encodings used to represent SOAP Remote Procedure Calls (RPC)).

The serialization rules defined in this section are identified by the URI "http://www.w3.org/2003/05/soap-encoding". SOAP messages using this particular serialization SHOULD indicate that fact by using the `SOAP encodingStyle` attribute

information item (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP encodingStyle Attribute](#)).

3.1 Mapping between XML and the SOAP Data Model

XML allows very flexible encoding of data. SOAP Encoding defines a narrower set of rules for encoding the graphs described in [2. SOAP Data Model](#). This section defines the encoding at a high level, and the subsequent sub-sections describe the encoding rules in more detail. The encodings described in this section can be used in conjunction with the mapping of RPC requests and responses specified in [4. SOAP RPC Representation](#).

The encodings are described below from the perspective of a de-serializer. In each case, the presence of an XML serialization is presumed, and the mapping to a corresponding graph is described.

More than one encoding is typically possible for a given graph. When serializing a graph for transmission inside a SOAP message, a representation that deserializes to the identical graph **MUST** be used; when multiple such representations are possible, any of them **MAY** be used. When receiving an encoded SOAP message, all representations **MUST** be accepted.

3.1.1 Encoding Graph Edges and Nodes

Each graph edge is encoded as an *element information item* and each *element information item* represents a graph edge. [3.1.3 Encoding Compound Values](#) describes the relationship between edge labels and the [local name] and [namespace name] properties of such *element information items*.

The graph node at which an edge terminates is determined by examination of the serialized XML as follows:

1. If the *element information item* representing the edge does not have a *ref attribute information item* (see [3.1.5.2 ref Attribute Information Item](#)) among its attributes then that *element information item* is said to *represent* a node in the graph and the edge terminates at that node. In such cases the *element information item* represents both a graph edge and a graph node
2. If the *element information item* representing the edge does have a *ref attribute information item* (see [3.1.5.2 ref Attribute Information Item](#)) among its attributes, then the value of that *attribute information item* **MUST** be identical to the value of exactly one *id attribute information item* (see [3.1.5.1 id Attribute Information Item](#)) in the same envelope. In this case the edge terminates at the graph node represented by the *element information item* on which the *id attribute information item* appears. That *element information item* **MUST** be in the scope of an *encodingStyle* attribute with a value of "http://www.w3.org/2003/05/soap-encoding" (see SOAP 1.2 Part 1 [\[SOAP Part 1\], SOAP encodingStyle Attribute](#)).

All nodes in the graph are encoded as described in 1 above. Additional inbound edges for

multi reference graph nodes are encoded as described in 2 above.

3.1.2 Encoding Simple Values

The lexical value of a graph node representing a simple value is the sequence of Unicode characters identified by the *character information item* children of the *element information item* representing that node. The *element information item* representing a simple value node MAY have among its attributes a 'nodeType' *attribute information item* (see [3.1.7 nodeType Attribute Information Item](#)). Note that certain Unicode characters cannot be represented in XML (see XML 1.0 [\[XML 1.0\]](#)).

3.1.3 Encoding Compound Values

An outbound edge of a graph node is encoded as an *element information item* child of the *element information item* that represents the node (see [3.1.1 Encoding Graph Edges and Nodes](#)). Particular rules apply depending on what kind of compound value the graph node represents. These rules are as follows:

1. For a graph edge which is distinguished by label, the [local name] and [namespace name] properties of the child *element information item* together determine the value of the edge label.
2. For a graph edge which is distinguished by position:
 - The ordinal position of the graph edge corresponds to the position of the child *element information item* relative to its siblings
 - The [local name] and [namespace name] properties of the child *element information item* are not significant.
3. The *element information item* representing a compound value node MAY have among its attributes a `nodeType` *attribute information item* (see [3.1.7 nodeType Attribute Information Item](#)).
4. The following rules apply to the encoding of a graph node that represents an "array":
 - The *element information item* representing an array node MAY have among its attributes an `itemType` *attribute information item* (see [3.1.4.1 itemType Attribute Information Item](#)).
 - The *element information item* representing an array node MAY have among its attributes an `arraySize` *attribute information item* (see [3.1.6 arraySize Attribute Information Item](#)).
5. If a graph edge does not terminate in a graph node then it can either be omitted from the serialization or it can be encoded as an *element information item* with an

xsi:nil attribute information item whose value is "true".

3.1.4 Computing the Type Name Property

The type name property of a graph node is a {namespace name, local name} pair computed as follows:

1. If the *element information item* representing the graph node has an `xsi:type` *attribute information item* among its attributes then the type name property of the graph node is the value of the `xsi:type` *attribute information item*.

Note:

This attribute is of type `xs:QName` (see XML Schema [\[XML Schema Part 2\]](#)); its value consists of the pair {namespace name, local name}. Neither the prefix used to construct the QName nor any information relating to any definition of the type is considered to be part of the value. The SOAP graph carries only the qualified name of the type.

2. Otherwise if the parent *element information item* of the *element information item* representing the graph node has an `enc:itemType` *attribute information item* (see [3.1.4.1 itemType Attribute Information Item](#)) among its attributes then the type name property of the graph node is the value of the `enc:itemType` *attribute information item*
3. Otherwise the value of the type name property of the graph node is unspecified.

Note:

These rules define how the type name property of a graph node in a graph is computed from a serialized encoding. This specification does not mandate validation using any particular schema language or type system. Nor does it include built in types or provide any standardized faults to reflect value/type name conflicts.

However, nothing prohibits development of additional specifications to describe the use of SOAP Encoding with particular schema languages or type systems. Such additional specifications MAY mandate validation using particular schema language, and MAY specify faults to be generated if validation fails. Such additional specifications MAY specify augmentations to the deserialized graph based on information determined from such a validation. The use by SOAP Encoding of `xsi:type` is intended to facilitate integration with the W3C XML Schema language (see [C. Using W3C XML Schema with SOAP Encoding](#)). Other XML based schema languages, data schemas and programmatic type systems MAY be used but only to the extent that they are compatible with the serialization described in this specification.

3.1.4.1 itemType Attribute Information Item

The `itemType` *attribute information item* has the following Infoset properties:

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

The type of the `itemType` *attribute information item* is `xs:QName`. The value of the `itemType` *attribute information item* is used to compute the type name property (see [3.1.4 Computing the Type Name Property](#)) of members of an array.

3.1.5 Unique identifiers

3.1.5.1 *id* Attribute Information Item

The `id` *attribute information item* has the following Infoset properties:

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

The type of the `id` *attribute information item* is `xs:ID`. The value of the `id` *attribute information item* is a unique identifier that can be referred to by a `ref` *attribute information item* (see [3.1.5.2 ref Attribute Information Item](#)).

3.1.5.2 *ref* Attribute Information Item

The `ref` *attribute information item* has the following Infoset properties:

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

The type of the `ref` *attribute information item* is `xs:IDREF`. The value of the `ref` *attribute information item* is a reference to a unique identifier defined by an `id` *attribute information item* (see [3.1.5.1 id Attribute Information Item](#)).

3.1.5.3 Constraints on *id* and *ref* Attribute Information Items

The value of a `ref` *attribute information item* MUST also be the value of exactly one `id`

attribute information item.

A `ref` *attribute information item* and an `id` *attribute information item* MUST NOT appear on the same *element information item*.

3.1.6 arraySize Attribute Information Item

The `arraySize` *attribute information item* has the following Infoset properties:

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

The type of the `arraySize` *attribute information item* is `enc:arraySize`. The value of the `arraySize` *attribute information item* MUST conform to the following EBNF grammar

```
[1] arraySizeValue ::= ("*" | concreteSize) nextConcreteSize*
[2] nextConcreteSize ::= whitespace concreteSize
[3] concreteSize ::= [0-9]+
[4] white space ::= (#x20 | #x9 | #xD | #xA)+
```

The array's dimensions are represented by each item in the list of sizes (unspecified size in case of the asterisk). The number of items in the list represents the number of dimensions in the array. The asterisk, if present, MUST only appear in the first position in the list. The default value of the `arraySize` *attribute information item* is "*", that is by default arrays are considered to have a single dimension of unspecified size.

3.1.7 nodeType Attribute Information Item

The `nodeType` *attribute information item* has the following Infoset properties:

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

The type of the `nodeType` *attribute information item* is `enc:nodeType`.

The value of the `nodeType` *attribute information item* MUST, if present, be one of the strings "simple" or "struct" or "array". The value indicates what kind of a value this node represents - a simple value, a compound struct value or a compound array value respectively.

3.2 Decoding Faults

During deserialization a SOAP receiver:

- SHOULD generate an "env:Sender" SOAP fault with a subcode of `enc:MissingID` if the message contains a `ref` *attribute information item* but no corresponding `id` *attribute information item* (see [3.1.5.3 Constraints on id and ref Attribute Information Items](#)).
- SHOULD generate an "env:Sender" SOAP fault with a subcode of `enc:DuplicateID` if the message contains two or more `id` *attribute information item* that have the same value. (see [3.1.5.3 Constraints on id and ref Attribute Information Items](#)).
- MAY generate an "env:Sender" SOAP fault with a subcode of `enc:UntypedValue` if the type name property of an encoded graph node is unspecified.

4. SOAP RPC Representation

One of the design goals of SOAP is to facilitate the exchange of messages that map conveniently to definitions and invocations of method and procedure calls in commonly used programming languages. For that purpose, this section defines a uniform representation of remote procedure call (RPC) requests and responses. It does not define actual mappings to any particular programming language. The representation is entirely platform independent and considerable effort has been made to encourage usage that is consistent with the Web in general.

As mentioned in section [2. SOAP Data Model](#), use and implementation of the SOAP RPC Representation is OPTIONAL.

The SOAP `encodingStyle` attribute information item (see SOAP 1.2 Part 1 [[SOAP Part 1](#)] [SOAP encodingStyle Attribute](#)) is used to indicate the encoding style of the RPC representation. The encoding thus specified MUST support the [2. SOAP Data Model](#). The encoding style defined in [3. SOAP Encoding](#) supports such constructs and is therefore suitable for use with the SOAP RPC Representation.

This SOAP RPC Representation is not predicated on any SOAP protocol binding. When SOAP is bound to HTTP, an RPC invocation maps naturally to an HTTP request and an RPC response maps to an HTTP response. (see [7. SOAP HTTP Binding](#)). However, the SOAP RPC Representation is not limited to the SOAP HTTP Binding.

To invoke an RPC, the following information is needed:

- The address of the target SOAP node.
- A procedure or method name.
- The identities and values of any arguments to be passed to the procedure or

method. Arguments used to identify Web resources SHOULD be distinguished from those representing data or control information (see [4.1.1 Identification of RPC Resources](#).)

- Values for properties as required by any features of the binding to be used. For example, "GET" or "POST" for the <http://www.w3.org/2003/05/soap/features/web-method/Method> property of the [6.4 SOAP Web Method Feature](#).
- Optional header data.

SOAP RPC relies on the protocol binding to provide a mechanism for carrying the URI of the target SOAP node. For HTTP the request URI indicates the resource against which the invocation is being made. Other than requiring it to be a valid URI, SOAP places no restriction on the form of an identifier (see RFC 2396 [[RFC 2396](#)] for more information on URIs). The section [4.1.1 Identification of RPC Resources](#) further discusses the use of URIs for identifying RPC resources.

The SOAP RPC Representation employs the [6.2 SOAP Request-Response Message Exchange Pattern](#) and [6.3 SOAP Response Message Exchange Pattern](#). Use of the SOAP RPC Representation with other MEPs MAY be possible, but is beyond the scope of this specification.

4.1 Use of RPC on the World Wide Web

The following guidelines SHOULD be followed when deploying SOAP RPC applications on the World Wide Web.

4.1.1 Identification of RPC Resources

The World Wide Web identifies resources with URIs, but common programming conventions convey identification information in the arguments to procedures, or in the names of those procedures. For example, the call:

```
updateQuantityInStock(PartNumber="123", NewQuantity="200")
```

suggests that the resource to be updated is the `QuantityInStock` for `PartNumber "123"`. Accordingly, when mapping to or from a programming language method or procedure call, any arguments that serve to identify resources (such as the part number above) should when practical be represented in the URI to which the SOAP message is addressed. When mapping to or from a programming language method or procedure call, the name of which identifies or qualifies the identification of a resource (such as `QuantityInStock` above), such naming or qualification should when practical be represented in the URI to which the SOAP message is addressed. No standard means of representation of arguments or method names is provided by this specification.

Note:

Conventions for specific URI encodings of procedure names and arguments, as well as for controlling the inclusion of such arguments in the SOAP RPC body could be established in conjunction with the development of Web Service interface description languages. They could be developed when SOAP is bound to particular programming languages or could be established on an application or procedure-specific basis.

4.1.2 Distinguishing Resource Retrievals from other RPCs

The World Wide Web depends on mechanisms that optimize commonly performed information retrieval tasks. Specifically, protocols such as HTTP [\[RFC 2616\]](#) provide a GET method which is used to perform safe retrievals, i.e. to perform retrievals that are idempotent, free of side effects, and for which security considerations do not preclude the use of cached results or URI-based resource identification.

Certain procedure or method calls represent requests for information retrieval. For example, the call:

```
getQuantityInStock(PartNumber="123")
```

might be used to retrieve the quantity established in the example above.

The following conventions can be employed to implement SOAP retrievals and other RPCs on the Web:

- The conventions described in [4.1.1 Identification of RPC Resources](#) are used to identify the resource with a URI.
- In cases where all the arguments have been represented in the URI, no SOAP header blocks are to be transmitted and the operation is a safe retrieval, the [6.4 SOAP Web Method Feature](#) and the [6.3 SOAP Response Message Exchange Pattern](#) are used. Accordingly, no SOAP envelope is transmitted for the request, and the `http://www.w3.org/2003/05/soap/features/web-method/Method` property is set to "GET". The results of the retrieval are a SOAP RPC response as described in [4.2.2 RPC Response](#)
- In cases where the operation to be performed is not a retrieval, when SOAP header blocks are to be transmitted (a digital signature, for example), or when a retrieval is not safe, the [6.4 SOAP Web Method Feature](#) and the [6.2 SOAP Request-Response Message Exchange Pattern](#) are used. The request envelope is encoded as described in [4.2.1 RPC Invocation](#), and the results are as described in [4.2.2 RPC Response](#). The `http://www.w3.org/2003/05/soap/features/web-method/Method` property is set to "POST".

The SOAP RPC Representation does not define any other value for the `http://www.w3.org/2003/05/soap/features/web-method/Method`.

4.2 RPC and SOAP Body

RPC invocations (except for safe retrievals: see [4.1.2 Distinguishing Resource Retrievals from other RPCs](#)) and responses are both carried in the SOAP `Body` element (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Body](#)) using the following representation:

4.2.1 RPC Invocation

An RPC invocation is modeled as follows:

- The invocation is represented by a single struct containing an outbound edge for each [in] or [in/out] parameter. The struct is named identically to the procedure or method name and the conventions of [B. Mapping Application Defined Names to XML Names](#) SHOULD be used to represent method names that are not legal XML names.
- Each outbound edge has a label corresponding to the name of the parameter. The conventions of [B. Mapping Application Defined Names to XML Names](#) SHOULD be used to represent parameter names that are not legal XML names.

Applications MAY process invocations with missing parameters but also MAY fail to process the invocation and return a fault.

4.2.2 RPC Response

An RPC response is modeled as follows:

- The response is represented by a single struct containing an outbound edge for the return value and each [out] or [in/out] parameter. The name of the struct is not significant.
- Each parameter is represented by an outbound edge with a label corresponding to the name of the parameter. The conventions of [B. Mapping Application Defined Names to XML Names](#) SHOULD be used to represent parameter names that are not legal XML names.
- A non-void return value is represented as follows:
 1. There MUST be an outbound edge with a local name of `result` and a namespace name of "http://www.w3.org/2003/05/soap-rpc" which terminates in a terminal node
 2. The type of that terminal node is a `xs:QName` and its value is the name of the outbound edge which terminates in the actual return value.

If the return value of the procedure is void then an outbound edge with a local name of `result` and a namespace name of "http://www.w3.org/2003/05/soap-rpc" MUST NOT be present.

- Invocation faults are handled according to the rules in [4.4 RPC Faults](#). If a protocol binding adds additional rules for fault expression, those MUST also be followed.

4.2.3 SOAP Encoding Restriction

When using SOAP encoding (see [3. SOAP Encoding](#)) in conjunction with the RPC convention described here, the SOAP `Body` MUST contain only a single child *element information item*, that child being the serialized RPC invocation or response struct.

4.3 RPC and SOAP Header

Additional information relevant to the encoding of an RPC invocation but not part of the formal procedure or method signature MAY be expressed in a SOAP envelope carrying an RPC invocation or response. Such additional information MUST be expressed as SOAP header blocks.

4.4 RPC Faults

The SOAP RPC Representation introduces additional SOAP fault subcode values to be used in conjunction with the fault codes described in SOAP 1.2 Part 1 [\[SOAP Part 1\]](#) [SOAP Fault Codes](#).

Errors arising during RPC invocations are reported according to the following rules:

1. A fault with a `value of Code` set to "env:Receiver" SHOULD be generated when the receiver cannot handle the message because of some temporary condition, e.g. when it is out of memory.

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 SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), [SOAP Code Element](#)).

2. A fault with a `value of Code` set to "env:DataEncodingUnknown" SHOULD be generated when the arguments are encoded in a data encoding unknown to the receiver.
3. A fault with a `value of Code` set to "env:Sender" and a `value of Subcode` set to "rpc:ProcedureNotPresent" MAY be generated when the receiver does not support the procedure or method specified.

Note:

Throughout this document, the term "value of Subcode " is used as a shorthand for "value of the `Value` child *element information item* of the `Subcode` *element information item*" (see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), [SOAP Subcode element](#)).

4. A fault with a `Value of Code` set to "env:Sender" and a `Value of Subcode` set to "rpc:BadArguments" MUST be generated when the receiver cannot parse the arguments or when there is a mismatch in number and/or type of the arguments between what the receiver expects and what was sent.
5. Other faults arising in an extension or from the application SHOULD be generated as described in SOAP 1.2 Part 1 [\[SOAP Part 1\]](#) [SOAP Fault Codes](#).

In all cases the values of the `Detail` and `Reason` *element information items* are implementation defined. Details of their use MAY be specified by an external document.

Note:

Senders might receive different faults from those listed above in response to an RPC invocation if the receiver does not support the (optional) RPC convention described here.

5. A Convention for Describing Features and Bindings

This section describes a convention describing Features (including MEPs) and Bindings in terms of properties and property values. The convention is sufficient to describe the distributed states of Feature and Binding specifications as mandated by the Binding Framework (see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#) [SOAP Protocol Binding Framework](#)) and it is used to describe a Request-Response MEP (see [6.2 SOAP Request-Response Message Exchange Pattern](#)), a Response MEP (see [6.3 SOAP Response Message Exchange Pattern](#)), the SOAP Web Method feature (see [6.4 SOAP Web Method Feature](#)) and the SOAP HTTP Binding (see [7. SOAP HTTP Binding](#)) elsewhere in this document. Along with the convention itself, an informal model is defined that describes how properties propagate through a SOAP system. Note that this model is intended to be illustrative only, and is not meant to imply any constraints on the structure or layering of any particular SOAP implementation.

5.1 Model and Properties

In general, a SOAP message is the information that one SOAP node wishes to exchange with another SOAP node according to a particular set of features, including a MEP. In addition, there may be information essential to exchanging a message that is not part of the message itself. Such information is sometimes called message metadata. In the

model, the message, any message metadata, and the various information items that enable features are represented as abstractions called properties.

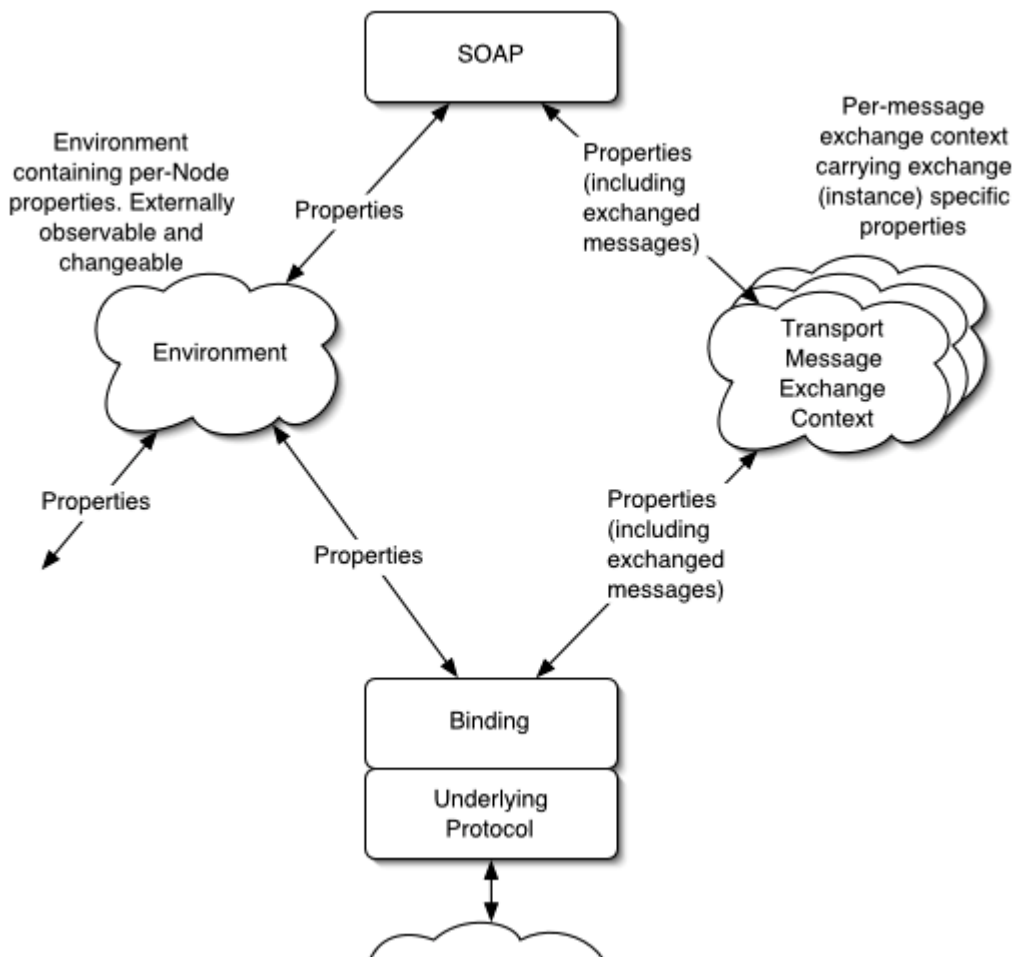
5.1.1 Properties

Under the convention, properties are represented as follows:

- Properties are named with URIs.
- Where appropriate, property values SHOULD have an XML Schema [\[XML Schema Part 1\]](#) [\[XML Schema Part 2\]](#) type listed in the specification which introduces the property.

5.1.2 Property Scope

Properties within a SOAP node differ in terms of their scope and the origins of their values. As shown in the figure below, we make the distinction between per message-exchange and more widely scoped properties by assigning them to different containers called Message Exchange Context and Environment Context respectively. All properties, regardless of their scope, are shared by a SOAP node and a particular Binding.



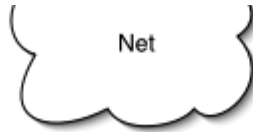


Figure 1: Model describing properties shared between SOAP and Binding

5.1.2.1 Message Exchange Context

A message exchange context is a collection of properties whose scope is limited to an instance of a given message exchange pattern. An example of a message exchange context property is the identifier of the message exchange pattern in use.

5.1.2.2 Environment Context

The environment context is a collection of properties whose scope extends beyond an instance of a given message exchange pattern. Examples of environment context properties are the IP address of the SOAP node or the current date and time.

The values of properties in Environment may depend upon local circumstances (as depicted by the external arrow from Environment in the figure above). More specifically, the properties in the example could be influenced by an operating system user ID on whose behalf a message exchange is being executed. The mapping of information in a particular implementation to such properties is outside the scope of the binding framework although the abstract representation of such information as properties is not.

5.1.3 Properties and Features

A feature may be expressed through multiple properties and a single property may enable more than one feature. For example, the properties called User ID and Password may be used to enable a feature called Authentication. As a second example, a single property called Message ID could be used to enable one feature called Transaction and a second feature called Message Correlation.

6. SOAP-Supplied Message Exchange Patterns and Features

6.1 Property Conventions for SOAP Message Exchange Patterns

Table 2 describes the properties (in accordance with the property naming conventions defined in this document) that support the description of message exchange patterns (MEPs). Other properties may be involved in the specification of particular MEPs, but the properties in this table are generally applicable to all MEPs.

Table 2: Property definitions supporting the description of MEPs

Property Name
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State>

6.2 SOAP Request-Response Message Exchange Pattern

This section defines the message exchange pattern (MEP) called "Request-Response". The description is an abstract presentation of the operation of this MEP. It is not intended to describe a real implementation or to suggest how a real implementation should be structured.

6.2.1 SOAP Feature Name

This message exchange pattern is identified by the URI (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Features](#)):

- "http://www.w3.org/2003/05/soap/mep/request-response/"

6.2.2 Description

The SOAP Request-Response MEP defines a pattern for the exchange of a SOAP message acting as a request followed by a SOAP message acting as a response. In the absence of failure in the underlying protocol, this MEP consists of exactly two SOAP messages.

In the normal operation of a message exchange conforming to the Request-Response MEP, a request message is first transferred from the requesting SOAP node to the responding SOAP node. Following the successful processing of the request message by the responding SOAP node, a response message is transferred from the responding SOAP node to the requesting SOAP node.

Abnormal operation during a Request-Response message exchange might be caused by a failure to transfer the request message, a failure at the responding SOAP node to process the request message, or a failure to transfer the response message. Such failures might be silent at either or both of the requesting and responding SOAP nodes involved, or might result in the generation of a SOAP or binding-specific fault (see [6.2.4 Fault Handling](#)). Also, during abnormal operation each SOAP node involved in the message exchange might differ in its determination of the successful completion of the message exchange.

The scope of a Request-Response MEP is limited to the exchange of a request message and a response message between one requesting and one responding SOAP node. This pattern does not mandate any correlation between multiple requests nor specific timing for multiple requests. Implementations MAY choose to support multiple ongoing requests (and associated response processing) at the same time.

6.2.3 State Machine Description

The Request-Response MEP defines a set of properties described in [Table 3](#).

Table 3: Property definitions for Request-Response MEP

Property Name	Property Description	Property Type
http://www.w3.org/2003/05/soap/mep/OutboundMessage	An abstract structure that represents the current outbound message in the message exchange. This abstracts both SOAP Envelope and any other information structures that are transferred along with the envelope.	Not specified
http://www.w3.org/2003/05/soap/mep/InboundMessage	An abstract structure that represents the current inbound message in the message exchange. This abstracts both SOAP Envelope	Not specified

	and any other information structures that are transferred along with the envelope.	
<code>http://www.w3.org/2003/05/soap/mep/ImmediateDestination</code>	The identifier of the immediate destination of an outbound message.	<code>xs:anyURI</code>
<code>http://www.w3.org/2003/05/soap/mep/ImmediateSender</code>	The identifier of the immediate sender of an inbound message.	<code>xs:anyURI</code>

To initiate a message exchange conforming to the Request-Response MEP, the requesting SOAP node instantiates a local message exchange context. [Table 4](#) describes how the context is initialized.

Table 4: Instar

Property Name
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State</code>
<code>http://www.w3.org/2003/05/soap/mep/OutboundMessage</code>
<code>http://www.w3.org/2003/05/soap/mep/ImmediateDestination</code>

There may be other properties related to the operation of the message exchange context instance. Such properties are initialized according to their own feature specifications.

Once the message exchange context is initialized, control of the context is passed to a (conforming) local binding instance.

The diagram below shows the logical state transitions at the requesting and responding SOAP nodes during the lifetime of the message exchange. At each SOAP node, the local binding instance updates (logically) the value of the

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State> property to reflect the current state of the message exchange. The state names are relative URIs, relative to a base URI value carried in the

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role> property of the local message exchange context.

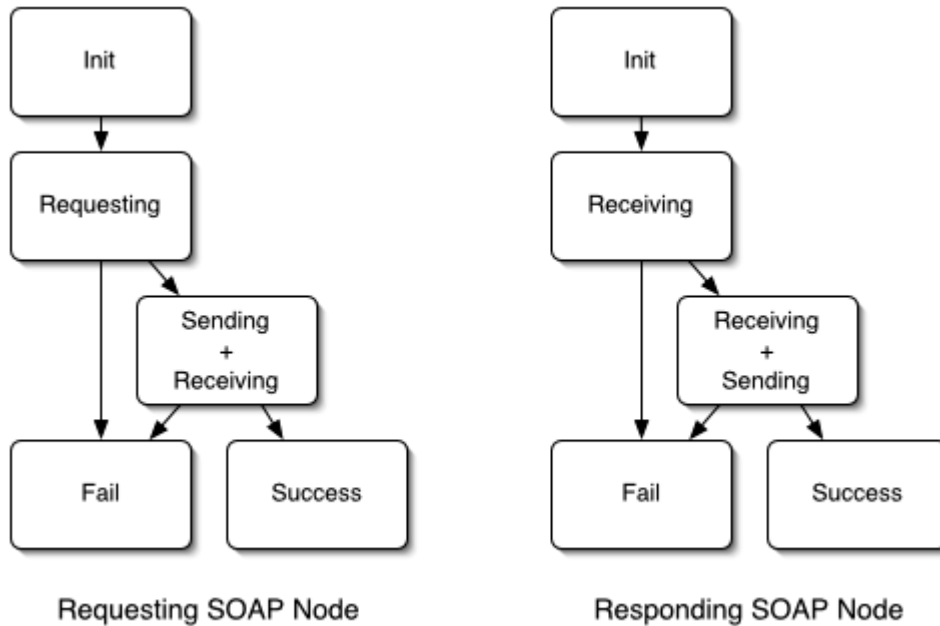


Figure 2: Request-Response MEP State Transition Diagram.

When the local binding instance at the responding SOAP node starts to receive an inbound request message, it (logically) instantiates a message exchange context. [Table 5](#) describes the properties that the binding initializes as part of the context's instantiation.

Table 5: Instantiation of Message Exchange Context

Property Name
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State>

When the requesting and responding SOAP nodes transition between states, the local binding instance (logically) updates a number of properties. [Table 6](#) and [Table 7](#) describe these updates for the requesting and the responding SOAP nodes, respectively.

Table 6: Requesting SOAP Node State Transitions

CurrentState	Transition Condition	NextState	
"Init"	Unconditional	"Requesting"	Initiate transmission of request message. http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State
"Requesting"	Message transmission failure	"Fail"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to "transmissionFailure"
	Start receiving response message	"Sending+Receiving"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to the response message (may differ from the response message abstraction of the response message). http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State
"Sending+Receiving"	Message exchange failure	"Fail"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to "exchangeFailure"
	Completed sending request message. Completed receiving response message.	"Success"	

Table 7: Responding SOAP Node State Transitions

CurrentState	Transition Condition	NextState
"Init"	Start receiving request message	"Receiving"

"Receiving"	Message reception failure	"Fail"
	Start of response message available in http://www.w3.org/2003/05/soap/mep/OutboundMessage	"Receiving+"
"Receiving+Sending"	Message exchange failure	"Fail"
	Completed receiving request message. Completed sending response message.	"Success"

Bindings that implement this MEP MAY provide for streaming of SOAP responses. That is, responding SOAP nodes MAY begin transmission of a SOAP response while a SOAP request is still being received and processed. When SOAP nodes implement bindings that support streaming, the following rules apply:

- All the rules in SOAP 1.2 Part 1 [\[SOAP Part 1\] Binding Framework](#) regarding streaming of individual SOAP messages MUST be obeyed for both request and response SOAP messages.
- When using streaming SOAP bindings, requesting SOAP nodes MUST avoid deadlock by accepting and if necessary processing SOAP response information while the SOAP request is being transmitted.

Note:

Depending on the implementation used and the size of the messages involved, this rule MAY require that SOAP applications stream application-level response processing in parallel with request generation.

- A requesting SOAP node MAY enter the "Fail" state, and thus abort transmission of the outbound SOAP request, based on information contained in an incoming streamed SOAP response.

6.2.4 Fault Handling

During the operation of the Request-Response MEP, the participating SOAP nodes may generate SOAP faults.

If a SOAP fault is generated by the responding SOAP node while it is in the "Receiving" state, the SOAP fault is made available in

<http://www.w3.org/2003/05/soap/mep/OutboundMessage> and the state machine transitions to the "Receiving+Sending" state.

This MEP makes no claims about the disposition or handling of SOAP faults generated by the requesting SOAP node during any processing of the response message that

follows the "Success" state in the requesting SOAP node's state transition table (see [Table 6](#)).

6.3 SOAP Response Message Exchange Pattern

This section defines the message exchange pattern (MEP) called "SOAP Response". The description is an abstract presentation of the operation of this MEP. It is not intended to describe a real implementation or to suggest how a real implementation should be structured.

6.3.1 SOAP Feature Name

This message exchange pattern is identified by the URI (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Features](#)):

- "http://www.w3.org/2003/05/soap/mep/soap-response/"

6.3.2 Description

The SOAP Response MEP defines a pattern for the exchange of a non-SOAP message acting as a request followed by a SOAP message acting as a response. In the absence of failure in the underlying protocol, this MEP consists of exactly two messages, only one of which is a SOAP message:

- A request transmitted in a binding-specific manner that does not include a SOAP envelope and hence does not involve any SOAP processing by the receiving SOAP node.
- A response message which contains a SOAP envelope. The MEP is completed by the processing of the SOAP envelope following the rules of the SOAP processing model (see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), section [SOAP Processing Model](#)).

Abnormal operation during a SOAP Response message exchange might be caused by a failure to transfer the request message or the response message. Such failures might be silent at either or both of the requesting and responding SOAP nodes involved, or might result in the generation of a SOAP or binding-specific fault (see section [6.3.4 Fault Handling](#)). Also, during abnormal operation each SOAP node involved in the message exchange might differ in its determination of the successful completion of the message exchange.

The scope of a SOAP Response MEP is limited to the request for an exchange of a response message between one requesting and one responding SOAP node. This pattern does not mandate any correlation between multiple requests nor specific timing for multiple requests. Implementations MAY choose to support multiple ongoing requests (and associated response processing) at the same time.

Note:

This MEP cannot be used in conjunction with features expressed as SOAP header blocks in the request because there is no SOAP envelope in which to carry them.

6.3.3 State Machine Description

The SOAP Response MEP defines a set of properties described in [Table 8](#).

Table 8: Property definitions for SOAP Response MEP

Property Name	Property Description	Property Type
http://www.w3.org/2003/05/soap/mep/OutboundMessage	An abstract structure that represents the current outbound message in the message exchange. This abstracts both SOAP Envelope Infoset (which MAY be null) and any other information structures that are transferred along with the envelope.	Not specified
http://www.w3.org/2003/05/soap/mep/InboundMessage	An abstract structure that represents the current inbound message in the message exchange. This abstracts both SOAP Envelope Infoset	Not specified

	(which MAY be null) and any other information structures that are transferred along with the envelope.	
<code>http://www.w3.org/2003/05/soap/mep/ImmediateDestination</code>	The identifier of the immediate destination of an outbound message.	<code>xs:anyURI</code>
<code>http://www.w3.org/2003/05/soap/mep/ImmediateSender</code>	The identifier of the immediate sender of an inbound message.	<code>xs:anyURI</code>

To initiate a message exchange conforming to the SOAP Response MEP, the requesting SOAP node instantiates a local message exchange context. [Table 9](#) describes how the context is initialized.

Table 9: Instanti

Property Name
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role</code>
<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State</code>
<code>http://www.w3.org/2003/05/soap/mep/OutboundMessage</code>
<code>http://www.w3.org/2003/05/soap/mep/ImmediateDestination</code>

There may be other properties related to the operation of the message exchange context instance. Such properties are initialized according to their own feature specifications.

Once the message exchange context is initialized, control of the context is passed to a (conforming) local binding instance.

The diagram below shows the logical state transitions at the requesting and responding SOAP nodes during the lifetime of the message exchange. At each SOAP node, the local binding instance updates (logically) the value of the

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State> property to reflect the current state of the message exchange. The state names are relative URIs, relative to a Base URI value carried in the

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role> property of the local message exchange context.

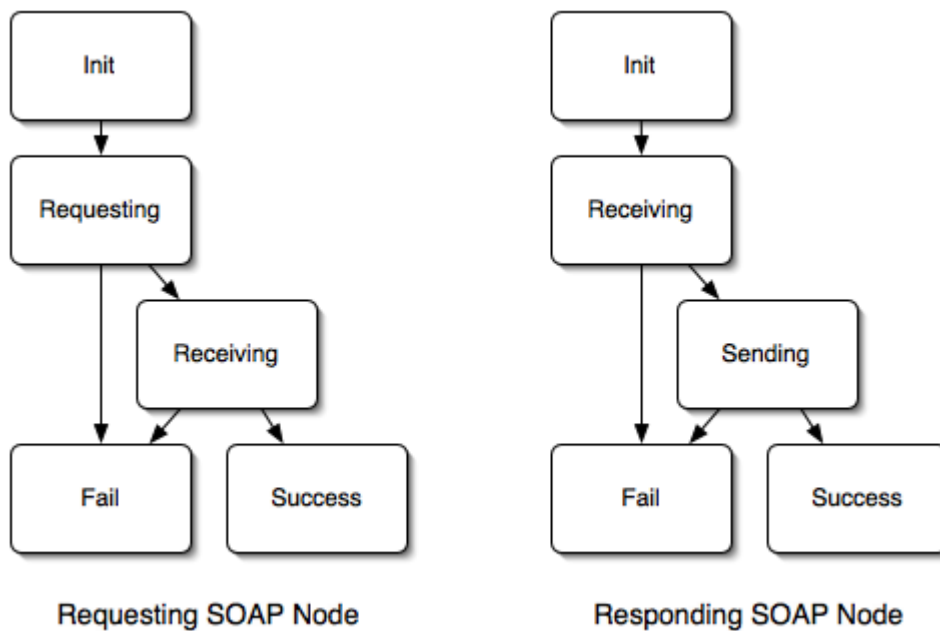


Figure 3: SOAP Response MEP State Transition Diagram

When the local binding instance at the responding SOAP node starts to receive an inbound request message, it (logically) instantiates a message exchange context. [Table 10](#) describes the properties that the binding initializes as part of the context's instantiation.

Table 10: Instanti

Property Name
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role>

<http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State>

When the requesting and responding SOAP nodes transition between states, the local binding instance (logically) updates a number of properties. [Table 11](#) and [Table 12](#) describe these updates for the requesting and the responding SOAP nodes, respectively.

Table 11: Requesting SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Unconditional	"Requesting"	Initiate request
"Requesting"	Message transmission failure	"Fail"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to "transmissionFailure"
	Start receiving response message	"Receiving"	Set http://www.w3.org/2003/05/soap/mep/ImmediateResponseMessage to the response message (may differ from the value of http://www.w3.org/2003/05/soap/mep/ImmediateResponseMessage); Set http://www.w3.org/2003/05/soap/mep/InboundMessage to an abstraction of the response message available in the context
"Receiving"	Message exchange failure	"Fail"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to "exchangeFailure"
	Completed receiving response message.	"Success"	

Table 12: Responding SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Start receiving request	"Receiving"	Set http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State to the request SOAP message

"Receiving"	Message reception failure	"Fail"	Set http:// to "rec
	Start of response message available in http://www.w3.org/2003/05/soap/mep/OutboundMessage	"Sending"	Initiate http://
"Sending"	Message exchange failure	"Fail"	Set http:// to "exc
	Completed sending response message.	"Success"	

6.3.4 Fault Handling

During the operation of the SOAP Response MEP, the participating SOAP nodes may generate SOAP faults.

If a SOAP fault is generated by the responding SOAP node while it is in the "Receiving" state, the SOAP fault is made available in

<http://www.w3.org/2003/05/soap/mep/OutboundMessage> and the state machine transitions to the "Sending" state.

This MEP makes no claims about the disposition or handling of SOAP faults generated by the requesting SOAP node during any processing of the response message that follows the "Success" state in the requesting SOAP node's state transition table (see [Table 11](#)).

6.4 SOAP Web Method Feature

This section defines the "SOAP Web Method Feature".

6.4.1 SOAP Feature Name

The SOAP Web Method feature is identified by the URI (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Features](#)):

- "<http://www.w3.org/2003/05/soap/features/web-method/>"

6.4.2 Description

Underlying protocols designed for use on the World Wide Web provide for manipulation of resources using a small set of Web methods such as GET, PUT, POST, and DELETE. These methods are formally defined in the HTTP specification [\[RFC 2616\]](#), but other underlying protocols might also support them. Bindings to HTTP or such other protocols SHOULD use the SOAP Web Method feature to give applications control over the Web methods to be used when sending a SOAP message.

Bindings supporting this feature SHOULD use the appropriate embodiment of that method if provided by the underlying protocol; for example, the HTTP binding provided with this specification represents the "GET" Web method as an HTTP GET request, and the "POST" method as an HTTP POST request (see [7. SOAP HTTP Binding](#)). Bindings supporting this feature SHOULD provide to the receiving node indication of the Web method used for transmission.

The SOAP Web Method feature MAY be implemented by bindings to underlying transports that have no preferred embodiment of particular Web methods (e.g. do not distinguish GET from POST). Such bindings SHOULD provide to the receiving node indication of the Web method used for transmission, but need take no other action in support of the feature.

6.4.3 SOAP Web Method Feature State Machine

The SOAP Web Method feature defines a single property, which is described in [Table 13](#).

Table 13: Property definition for the SOAP Web Method feature

Property Name	Property Description	Property Type
http://www.w3.org/2003/05/soap/features/web-method/Method	One of "GET", "POST", "PUT", "DELETE" (or others which may subsequently be added to the repertoire of Web methods.)	Not specified

This specification provides for the use of the SOAP Web Method feature in conjunction with the [6.2 SOAP Request-Response Message Exchange Pattern](#) and [6.3 SOAP Response Message Exchange Pattern](#) message exchange patterns. This feature MAY be used with other MEPs if and only if provided for in the specifications of those MEPs.

A node sending a request message MUST provide a value for the <http://www.w3.org/2003/05/soap/features/web-method/Method> property. A protocol binding supporting this feature SHOULD set the value of the <http://www.w3.org/2003/05/soap/features/web-method/Method> property at the receiving node to match that provided by the sender; the means of transmission for the method property is binding-specific.

A responding node SHOULD respond in a manner consistent with the Web method requested (e.g. a "GET" should result in retrieval of a representation of the identified resource) or SHOULD fault in an application-specific manner if the Web method cannot be supported.

Bindings implementing this feature MUST employ a Message Exchange Pattern with

semantics that are compatible with the Web method selected. For example, the SOAP Response Message Exchange Pattern (see [6.3 SOAP Response Message Exchange Pattern](#)) is compatible with GET.

6.5 SOAP Action Feature

This section defines the "SOAP Action Feature".

6.5.1 SOAP Feature Name

The SOAP Action feature is identified by the URI (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Features](#)):

- "http://www.w3.org/2003/05/soap/features/action/"

6.5.2 Description

Many SOAP 1.2 underlying protocol bindings will likely utilize the "application/soap+xml" media type (described in [A. The application/soap+xml Media Type](#)) to transmit XML serializations of SOAP messages. The media type specifies an optional `action` parameter (see [A.3 The action Parameter](#)), which can be used to optimize dispatch or routing, among other things. The Action Feature specifies well-known URIs to indicate support for the `action` parameter in bindings which use MIME, and also to refer to value of the parameter itself.

6.5.3 SOAP Action Feature State Machine

The SOAP Action feature defines a single property, which is described in [Table 14](#).

Table 14: Property definition for the SOAP Action feature

Property Name	Property Type
<code>http://www.w3.org/2003/05/soap/features/action/Action</code>	<code>xsd:anyURI</code>

If the `http://www.w3.org/2003/05/soap/features/action/Action` property has a value at a SOAP sender utilizing a binding supporting this feature, the sender MUST use the property value as the value of the `action` parameter in the media type designator.

Conversely, if a value arrives in the `action` parameter of the media type designator at a SOAP receiver, the receiver MUST make that value available as the value of the `http://www.w3.org/2003/05/soap/features/action/Action` property.

7. SOAP HTTP Binding

7.1 Introduction

The SOAP HTTP Binding provides a binding of SOAP to HTTP. The binding conforms to the SOAP Protocol Binding Framework (see SOAP 1.2 Part 1 [[SOAP Part 1](#)] [SOAP Protocol Binding Framework](#)) and supports the message exchange patterns and features described in [6. SOAP-Supplied Message Exchange Patterns and Features](#).

7.1.1 Optionality

The SOAP HTTP Binding is optional and SOAP nodes are NOT required to implement it. A SOAP node that correctly and completely implements the SOAP HTTP Binding may be said to "conform to the SOAP 1.2 HTTP Binding."

The SOAP version 1.2 specification does not preclude development of other bindings to HTTP or bindings to other protocols, but communication with nodes using such other bindings is not a goal. Note that other bindings of SOAP to HTTP MAY be written to provide support for SOAP Message exchange patterns other than [6.2 SOAP Request-Response Message Exchange Pattern](#) or the [6.3 SOAP Response Message Exchange Pattern](#). Such alternate bindings MAY therefore make use of HTTP features and status codes not required for this binding. For example, another binding might provide for a 202 or 204 HTTP response status to be returned in response to an HTTP POST or PUT (e.g. a one-way "push" MEP with confirmation).

7.1.2 Use of HTTP

The SOAP HTTP binding defines a base URI according to the rules in HTTP/1.1 [[RFC 2616](#)]. I.e. the base URI is the HTTP Request-URI or the value of the HTTP Content-Location header field.

This binding of SOAP to HTTP is intended to make appropriate use of HTTP as an application protocol. For example, successful responses are sent with status code 200, and failures are indicated as 4XX or 5XX. This binding is not intended to fully exploit the features of HTTP, but rather to use HTTP specifically for the purpose of communicating with other SOAP nodes implementing the same binding. Therefore, this HTTP binding for SOAP does not specify the use and/or meaning of all possible HTTP methods, header fields and status responses. It specifies only those which are pertinent to the [6.2 SOAP Request-Response Message Exchange Pattern](#) or the [6.3 SOAP Response Message Exchange Pattern](#), or which are likely to be introduced by HTTP mechanisms (such as proxies) acting between the SOAP nodes.

Certain optional features provided by this binding depend on capabilities provided by HTTP/1.1, for example content negotiation. Implementations SHOULD thus use HTTP/1.1 [[RFC 2616](#)] (or later compatible versions that share the same major version number). Implementations MAY also be deployed using HTTP/1.0, although in this case certain optional binding features may not be provided.

Note:

SOAP HTTP Binding implementations need to account for the fact that HTTP/1.0 intermediaries (which may or may not also be SOAP intermediaries) may alter the representation of SOAP messages, even in situations where both the initial SOAP sender and ultimate SOAP receiver use HTTP/1.1.

7.1.3 Interoperability with non-SOAP HTTP Implementations

Particularly when used with the [6.3 SOAP Response Message Exchange Pattern](#), the HTTP messages produced by this binding are likely to be indistinguishable from those produced by non-SOAP implementations performing similar operations. Accordingly, some degree of interoperability can be made possible between SOAP nodes and other HTTP implementations when using this binding. For example, a conventional Web server (i.e. one not written specifically to conform to this specification) might be used to respond to SOAP-initiated HTTP GET's with representations of `Content-Type` "application/soap+xml". Such interoperability is not a normative feature of this specification.

Even though HTTP often is used on the well-known TCP port 80, the use of HTTP is not limited to that port. As a result, it is possible to have a dedicated HTTP server for handling SOAP processing on a distinct TCP port. Alternatively, it is possible to use a separate virtual host for dealing with SOAP processing. Such configuration, however, is a matter of convenience and is not a requirement of this specification (see SOAP 1.2 Part 1 [\[SOAP Part 1\] Binding to Application-Specific Protocols](#)).

7.1.4 HTTP Media-Type

Conforming implementations of this binding:

1. MUST be capable of sending and receiving messages serialized using media type "application/soap+xml" whose proper use and parameters are described in [A. The application/soap+xml Media Type](#).
2. MAY send requests and responses using other media types providing that such media types provide for at least the transfer of SOAP XML Infoset.
3. MAY, when sending requests, provide an HTTP Accept header field. This header field:
 - SHOULD indicate an ability to accept at minimum "application/soap+xml".
 - MAY additionally indicate willingness to accept other media types that satisfy 2 above.

7.2 Binding Name

This binding is identified by the URI (see SOAP 1.2 Part 1 [\[SOAP Part 1\] SOAP Protocol](#))

[Binding Framework](#)):

- "http://www.w3.org/2003/05/soap/bindings/HTTP/"

7.3 Supported Message Exchange Patterns

An implementation of the SOAP HTTP Binding MUST support the following message exchange patterns (MEPs):

- "http://www.w3.org/2003/05/soap/mep/request-response/" (see [6.2 SOAP Request-Response Message Exchange Pattern](#))
- "http://www.w3.org/2003/05/soap/mep/soap-response/" (see [6.3 SOAP Response Message Exchange Pattern](#))

7.4 Supported Features

An implementation of the SOAP HTTP Binding MUST support the following features:

- "http://www.w3.org/2003/05/soap/features/web-method/" (see [6.4 SOAP Web Method Feature](#))
- "http://www.w3.org/2003/05/soap/features/action/" (see [6.5 SOAP Action Feature](#))

The possible values of `http://www.w3.org/2003/05/soap/features/web-method/Method` property are restricted in this HTTP binding according to the MEP in use (as present in `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`

Table 15: Possible values of the Web-Method Method

<code>http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName</code>
<code>"http://www.w3.org/2003/05/soap/mep/request-response/"</code>
<code>"http://www.w3.org/2003/05/soap/mep/soap-response/"</code>

Note:

other SOAP Version 1.2 bindings to HTTP may permit other combinations of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName` and `http://www.w3.org/2003/05/soap/features/web-method/Method`.

7.5 MEP Operation

For binding instances conforming to this specification:

- A SOAP node instantiated at an HTTP client may assume the role (i.e. the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`) of "RequestingSOAPNode".
- A SOAP node instantiated at an HTTP server may assume the role (i.e. the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`) of "RespondingSOAPNode".

The remainder of this section describes the MEP state machine and its relation to the HTTP protocol. In the state tables below, the states are defined as values of the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State` (see [6.2 SOAP Request-Response Message Exchange Pattern](#) and [6.3 SOAP Response Message Exchange Pattern](#)), and are of type `xs:anyURI` . For brevity, relative URIs are used, the base URI being the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role` .

The message exchange pattern in use is indicated by the HTTP method used in the request. HTTP GET corresponds to the SOAP-Response MEP, HTTP POST corresponds to the SOAP Request-Response MEP.

7.5.1 Behavior of Requesting SOAP Node

The overall flow of the behavior of a requesting SOAP node follows a state machine description consistent with either [6.2 SOAP Request-Response Message Exchange Pattern](#) or [6.3 SOAP Response Message Exchange Pattern](#) (differences are indicated as necessary.) This binding supports streaming and, as a result, requesting SOAP nodes MUST avoid deadlock by accepting and if necessary processing SOAP response information while the SOAP request is being transmitted (see [6.2.3 State Machine Description](#)). The following subsections describe each state in detail.

7.5.1.1 Init

In the "Init" state, a HTTP request is formulated according to [Table 16](#) and transmission of the request is initiated.

Table 16: HTTP Request Fields

Field	Value
HTTP Method	According to the <code>http://www.w3.org/2003/05/soap/features/web-method/Method</code> property. POST and GET are the only values supported by this binding.
Request URI	The value of the URI carried in the <code>http://www.w3.org/2003/05/soap/mep/ImmediateDestination</code> property of the message exchange context.

Content-Type header field	The media type of the request entity body (if present) otherwise, omitted (see 7.1 Introduction for a description of permissible media types). If the SOAP envelope info:et in the <code>http://www.w3.org/2003/05/soap/mep/OutboundMessage</code> property is null, then the Content-Type header field MAY be omitted.
action parameter	According to the value of the <code>http://www.w3.org/2003/05/soap/features/action/Action</code> property.
Accept header field (optional)	List of media types that are acceptable in response to the request message.
Additional header fields	Generated in accordance with the rules for the binding specific expression of any optional features in use for this message exchange. For example, a Content-Encoding header field (see HTTP RFC 2616 , section 14.11) may be used to express an optional compression feature.
HTTP entity body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-Type header field. Rules for carrying SOAP messages in media type "application/soap+xml" are given in A. The application/soap+xml Media Type . If the SOAP envelope info:et in the <code>http://www.w3.org/2003/05/soap/mep/OutboundMessage</code> property is null, the entity body is omitted

7.5.1.2 Requesting

In the "Requesting" state, sending of the request continues while waiting for the start of the response message. [Table 17](#) details the transitions that take place when a requesting SOAP node receives an HTTP status line and response header fields. For some status codes there is a choice of possible next state. In cases where "Fail" is one of the choices, the transition is dependent on whether a SOAP message is present in the HTTP response. If a SOAP message is present, the next state is "Sending+Receiving" or "Receiving", otherwise the next state is "Fail". The choice between "Sending+Receiving" and "Receiving" depends of the MEP in use: "Sending+Receiving" is the next state for Request-Response while "Receiving" is the next state for SOAP-Response.

Table 17: HTTP status code dependent transitions

Status Code	Reason phrase	Significance/Action	
2xx	Successful		
200	OK	The response message follows in HTTP response entity body. Start making an abstraction of the response message available in <code>http://www.w3.org/2003/05/soap/mep/InboundMessage</code> .	"Send or "Re

3xx	Redirection	The requested resource has moved and the HTTP request SHOULD be retried using the URI carried in the associated Location header field as the new value for the http://www.w3.org/2003/05/soap/mep/ImmediateDestination property.	"Init"
4xx	Client Error		
400	Bad Request	Indicates a problem with the received HTTP request message.	"Send "Rece
401	Unauthorized	Indicates that the HTTP request requires authorization. If the simple authentication feature is unavailable or the operation of simple authentication ultimately fails, then the message exchange is regarded as having completed unsuccessfully.	"Requ
405	Method not allowed	Indicates that the peer HTTP server does not support the requested HTTP method at the given request URI. The message exchange is regarded as having completed unsuccessfully.	"Fail"
415	Unsupported Media Type	Indicates that the peer HTTP server does not support the Content-type used to encode the request message. The message exchange is regarded as having completed unsuccessfully.	"Fail"
5xx	Server Error		
500	Internal Server Error	Indicates a server problem or a problem with the received request	"Send "Rece

Table 17 refers to some but not all of the existing HTTP/1.1 [\[RFC 2616\]](#) status codes. In addition to these status codes, HTTP provides an open-ended mechanism for supporting status codes defined by HTTP extensions (see RFC 2817 [\[RFC 2817\]](#) for a registration mechanism for new status codes). HTTP status codes are divided into status code classes as described in HTTP [\[RFC 2616\]](#), section 6.1.1. The SOAP HTTP binding follows the rules of any HTTP application which means that an implementation of the SOAP HTTP binding must understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response must not be cached.

Note:

There may be elements in the HTTP infrastructure configured to modify HTTP response entity bodies for 4xx and 5xx status code responses. For example, some HTTP origin servers have such a feature as a configuration option. This behavior may interfere with the use of 4xx and 5xx status code responses carrying SOAP fault messages in HTTP and it is recommended that such behavior is disabled for resources accepting SOAP/HTTP requests. If the rewriting behavior cannot be

disabled, SOAP/HTTP cannot be used in such configurations.

7.5.1.3 *Sending+Receiving*

In the "Sending+Receiving" state ([6.2 SOAP Request-Response Message Exchange Pattern](#) only), the transmission of the request message and receiving of the response message is completed. The response message is assumed to contain a SOAP envelope serialized according to the rules for carrying SOAP messages in the media type given in the Content-Type header field.

The response MAY be of content type other than "application/soap+xml". Such usage is considered non-normative, and accordingly is not modeled in the state machine. Interpretation of such responses is at the discretion of the receiver.

7.5.1.4 *Receiving*

In the "Receiving" state ([6.3 SOAP Response Message Exchange Pattern](#) only), receiving of the response message is completed. The response message is assumed to contain a SOAP envelope serialized according to the rules for carrying SOAP messages in the media type given in the Content-Type header field.

The response MAY be of content type other than "application/soap+xml". Such a result is particularly likely when a SOAP request sent with a `http://www.w3.org/2003/05/soap/features/web-method/Method` of "GET" is directed (intentionally or otherwise) to a non-SOAP HTTP server. Such usage is considered non-normative, and accordingly is not modeled in the state machine. Interpretation of such responses is at the discretion of the receiver.

7.5.1.5 *Success and Fail*

"Success" and "Fail" are the terminal states of the Request-Response and SOAP-Response MEPs. Control over the message exchange context returns to the local SOAP node.

7.5.2 Behavior of Responding SOAP Node

The overall flow of the behavior of a responding SOAP node follows a state machine description consistent with either [6.2 SOAP Request-Response Message Exchange Pattern](#) or [6.3 SOAP Response Message Exchange Pattern](#) (differences are indicated as necessary). The following subsections describe each state in detail.

7.5.2.1 *Init*

In the "Init" state, the binding waits for the start of an inbound request message. [Table 18](#) describes the errors that a responding SOAP node might generate while in the "Init" state. In this state no SOAP message has been received, therefore the SOAP node cannot generate a SOAP fault.

Table 18: Errors generated in the Init state

Problem with Message	HTTP Status Code	HTTP Reason Phrase (informative)
Malformed Request Message	400	Bad request
HTTP Method is neither POST nor GET	405	Method Not Allowed
Unsupported message encapsulation method	415	Unsupported Media

7.5.2.2 Receiving

In the "Receiving" state, the binding receives the request and any associated message and waits for the start of a response message to be available. [Table 19](#) describes the HTTP response header fields generated by the responding SOAP node. [Table 20](#) describes the HTTP status codes associated with SOAP faults that can be generated by the responding SOAP node.

Table 19: HTTP Response Headers Fields

Field	Value
Status line	200, or set according to Table 20 if a SOAP fault was generated.
Content-Type header field	The media type of the response body, see 7.1 Introduction for a description of permissible media types.
Additional header fields	Generated in accordance with the rules for the binding specific expression of any optional features in use for this message exchange. For example, a Content-Encoding header field (see HTTP [RFC 2616] , section 14.11) may be used to express an optional compression feature.
HTTP Entity Body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-Type header field. Rules for carrying SOAP messages in "application/soap+xml" are given in A. The application/soap+xml Media Type .

Table 20: SOAP Fault to HTTP Status Mapping

SOAP Fault	HTTP Status Code	HTTP Reason Phrase (informative)
env:VersionMismatch	500	Internal server error
env:MustUnderstand	500	Internal server error
env:Sender	400	Bad request
env:Receiver	500	Internal server error

env:DataEncodingUnknown	500	Internal server error
-------------------------	-----	-----------------------

7.5.2.3 Receiving+Sending

In the "Receiving+Sending" state ([6.2 SOAP Request-Response Message Exchange Pattern](#) only) the binding completes receiving of the request message and transmission of the response message.

7.5.2.4 Sending

In the "Sending" state ([6.3 SOAP Response Message Exchange Pattern](#) only) the binding completes transmission of the response message.

7.5.2.5 Success and Fail

"Success" and "Fail" are the terminal states for the Request-Response and SOAP-Response MEPs. From the point-of-view of the local node this message exchange has completed.

7.6 Security Considerations

The SOAP HTTP Binding (see [7. SOAP HTTP Binding](#)) can be considered as an extension of the HTTP application protocol. As such, all of the security considerations identified and described in section 15 of the HTTP specification [[RFC 2616](#)] apply to the SOAP HTTP Binding in addition to those described in SOAP 1.2 Part 1 [[SOAP Part 1 Security Considerations](#)]. Implementors of the SOAP HTTP Binding should carefully review this material.

8. References

8.1 Normative References

[SOAP Part 1]

W3C Proposed Recommendation "SOAP Version 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 24 June 2003 (See <http://www.w3.org/TR/2003/REC-soap12-part1-20030624>.)

[RFC 2616]

IETF "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, January 1997. (See <http://www.ietf.org/rfc/rfc2616.txt>.)

[RFC 2119]

IETF "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. (See <http://www.ietf.org/rfc/rfc2119.txt>.)

[XML Schema Part 1]

W3C Recommendation "XML Schema Part 1: Structures", Henry S. Thompson,

David Beech, Murray Maloney, Noah Mendelsohn, 2 May 2001. (See [http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.](http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/))

[XML Schema Part 2]

W3C Recommendation "XML Schema Part 2: Datatypes", Paul V. Biron, Ashok Malhotra, 2 May 2001. (See [http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.](http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/))

[RFC 2396]

IETF "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. (See <http://www.ietf.org/rfc/rfc2396.txt>.)

[Namespaces in XML]

W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See [http://www.w3.org/TR/1999/REC-xml-names-19990114/.](http://www.w3.org/TR/1999/REC-xml-names-19990114/))

[XML 1.0]

W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)", Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 6 October 2000. (See [http://www.w3.org/TR/2000/REC-xml-20001006/.](http://www.w3.org/TR/2000/REC-xml-20001006/))

[XML InfoSet]

W3C Recommendation "XML Information Set", John Cowan, Richard Tobin, 24 October 2001. (See [http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.](http://www.w3.org/TR/2001/REC-xml-infoset-20011024/))

[RFC 3023]

IETF "RFC 3023: XML Media Types", M. Murata, S. St. Laurent, D. Kohn, July 1998. (See <http://www.ietf.org/rfc/rfc3023.txt>.)

8.2 Informative References

[SOAP Part 0]

W3C Proposed Recommendation "SOAP Version 1.2 Part 0: Primer", Nilo Mitra, 24 June 2003 (See [http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.](http://www.w3.org/TR/2003/REC-soap12-part0-20030624/))

[SOAP MediaType]

IETF Internet Draft "The 'application/soap+xml' media type", M. Baker, M. Nottingham, "draft-baker-soap-media-reg-03.txt" May 29, 2003. Note that this Internet Draft expires in November 2003. (See <http://www.ietf.org/internet-drafts/draft-baker-soap-media-reg-03.txt>.)

[XMLP Comments]

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

[XMLP Charter]

XML Protocol Charter (See <http://www.w3.org/2000/09/XML-Protocol-Charter.>)

[RFC 2045]

IETF "RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, N. Borenstein, November 1996. (See <http://www.ietf.org/rfc/rfc2045.txt>.)

[RFC 2026]

IETF "RFC 2026: The Internet Standards Process -- Revision 3", section 4.2.3, S. Bradner, October 1996. (See <http://www.ietf.org/rfc/rfc2026.txt>.)

[RFC 2817]

IETF "RFC 2817: Upgrading to TLS Within HTTP/1.1", R. Khare, S. Lawrence, May 2000. (See <http://www.ietf.org/rfc/rfc2817.txt>.)

[CharMod]

W3C Working Draft "Character Model for the World Wide Web 1.0", Martin J. Durst, Francois Yergeau, Richard Ishida, Misha Wolf, Asmus Freytag, Tex Texin, 30 April 2002. (See <http://www.w3.org/TR/charmod/>.)

A. The "application/soap+xml" Media Type

This appendix defines the "application/soap+xml" media type which can be used to describe SOAP 1.2 messages serialized as XML.

Note:

At the time of this document's publication, the process for registering this media type with IANA was underway using the [\[SOAP MediaType\]](#) Internet Draft. A future version of the SOAP specification may reference the media type in the IANA registry.

A.1 Registration

MIME media type name:

application

MIME subtype name:

soap+xml

Required parameters:

none

Optional parameters:

charset

This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in RFC 3023 [\[RFC 3023\]](#).

action

See section [A.3 The action Parameter](#).

Encoding considerations:

Identical to those of "application/xml" as described in RFC 3023 [\[RFC 3023\]](#), section 3.2, as applied to the SOAP envelope infoset.

Security considerations:

See section [A.2 Security Considerations](#).

Interoperability considerations:

There are no known interoperability issues.

Published specification:

This document (SOAP 1.2 Part 2) and SOAP 1.2 Part 1 [\[SOAP Part 1\]](#).

Applications which use this media type:

No known applications currently use this media type.

Additional information:

File extension:

SOAP messages are not required or expected to be stored as files.

Fragment identifiers:

Identical to that of "application/xml" as described in RFC 3023 [\[RFC 3023\]](#), section 5.

Base URI:

As specified in RFC 3023 [\[RFC 3023\]](#), section 6. Also see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), section [Use of URIs in SOAP](#).

Macintosh File Type code:

TEXT

Person and email address to contact for further information:

Mark Baker <mbaker@idokorro.com>

Intended usage:

COMMON

Author/Change controller:

The SOAP 1.2 specification set is a work product of the World Wide Web Consortium's [XML Protocol Working Group](#). The W3C has change control over these specifications.

A.2 Security Considerations

Because SOAP can carry application defined data whose semantics is independent from that of any MIME wrapper (or context within which the MIME wrapper is used), one should not expect to be able to understand the semantics of the SOAP message based on the semantics of the MIME wrapper alone. Therefore, whenever using the "application/soap+xml" media type, it is strongly RECOMMENDED that the security implications of the context within which the SOAP message is used is fully understood. The security implications are likely to involve both the specific SOAP binding to an underlying protocol as well as the application-defined semantics of the data carried in the SOAP message (though one must be careful when doing this, as discussed in SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), section [Binding to Application-Specific Protocols](#).

Also, see SOAP 1.2 Part 1 [\[SOAP Part 1\]](#), the entire section [Security Considerations](#).

In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in RFC 3023 [\[RFC 3023\]](#), section 10.

A.3 The `action` Parameter

This optional parameter can be used to specify the URI that identifies the intent of the message. In SOAP 1.2, it serves a similar purpose as the `SOAPAction` HTTP header field did in SOAP 1.1. Namely, its value identifies the intent of the message.

The value of the action parameter is an absolute URI-reference as defined by RFC 2396 [\[RFC 2396\]](#). SOAP places no restrictions on the specificity of the URI or that it is resolvable.

Although the purpose of the action parameter is to indicate the intent of the SOAP message there is no mechanism for automatically computing the value based on the SOAP envelope. In other words, the value has to be determined out of band.

It is recommended that the same value be used to identify sets of message types that are logically connected in some manner, for example part of the same "service". It is strongly RECOMMENDED that the URI be globally unique and stable over time.

The presence and content of the action parameter MAY be used by servers such as firewalls to appropriately filter SOAP messages and it may be used by servers to facilitate dispatching of SOAP messages to internal message handlers etc. It SHOULD NOT be used as an insecure form of access authorization.

Use of the action parameter is OPTIONAL. SOAP Receivers MAY use it as a hint to optimize processing, but SHOULD NOT require its presence in order to operate.

B. Mapping Application Defined Names to XML Names

This appendix details an algorithm for taking an application defined name, such as the name of a variable or field in a programming language, and mapping it to the Unicode characters that are legal in the names of XML elements and attributes as defined in Namespace in XML [[Namespaces in XML](#)]

Hex Digits

[5] `hexDigit ::= [0-9A-F]`

B.1 Rules for Mapping Application Defined Names to XML Names

1. An XML Name has two parts: *Prefix* and *LocalPart*. Let *Prefix* be determined per the rules and constraints specified in Namespaces in XML [[Namespaces in XML](#)].
2. Let *T* be a name in an application, represented as a sequence of characters encoded in a particular character encoding.
3. Let M be the implementation-defined function for transcoding of the characters used in the application defined name to an equivalent string of Unicode characters.

Note:

Ideally, if this transcoding is from a non-Unicode encoding, it should be both reversible and Unicode Form C normalizing (that is, combining sequences will be in the prescribed canonical order). It should be noted that some transcodings cannot be perfectly reversible and that Normalization Form C (NFC) normalization may alter the original sequence in a few cases (see Character Model for the World Wide Web [[CharMod](#)]). To ensure that matching names continue to match after mapping, Unicode sequences should be normalized using Unicode Normalization Form C.

Note:

This transcoding is explicitly to Unicode scalar values ("code points") and not to any particular character encoding scheme of Unicode, such as UTF-8 or UTF-16.

Note:

Note: Properly formed surrogate pair sequences must be converted to their

respective scalar values ("code points") [That is, the sequence U+D800 U+DC00 should be transcoded to the character U+10000]. If the transcoding begins with a Unicode encoding, non-conforming (non-shortest form) UTF-8 and UTF-16 sequences must be converted to their respective scalar values.

Note:

The number of characters in T is not necessarily the same as the number of characters in M , because transcoding may be one-to-many or many-to-one. The details of transcoding may be implementation-defined. There may be (very rarely) cases where there is no equivalent Unicode representation for T ; such cases are not covered here.

4. Let C be the sequence of Unicode scalar values (characters) represented by $M(T)$
5. Let N be the number of characters in C . Let C_1, C_2, \dots, C_N be the characters of C , in order from most to least significant (logical order).
6. For each i between 1 (one) and N , let X_i be the Unicode character string defined by the following rules:

Case:

1. If C_i is undefined (that is, some character or sequence of characters as defined in the application's character sequence T contains no mapping to Unicode), then X_i is implementation-defined.
2. If $i \leq N-1$ and C_i is "_" (U+005F LOW LINE) and C_{i+1} is "x" (U+0078 LATIN SMALL LETTER X), then let X_i be "_x005F_".
3. If $i=1$, and $N \geq 3$, and C_1 is "x" (U+0078 LATIN SMALL LETTER X) or "X" (U+0058 LATIN CAPITAL LETTER X), and C_2 is "m" (U+006D LATIN SMALL LETTER M) or "M" (U+004D LATIN CAPITAL LETTER M), and C_3 is "l" (U+006C LATIN SMALL LETTER L) or "L" (U+004C LATIN CAPITAL LETTER L) (in other words, a string three letters or longer starting with the text "xml" or any re-capitalization thereof), then if C_1 is "x" (U+0078 LATIN SMALL LETTER X) then let X_1 be "_x0078_"; otherwise, if C_1 is "X" (U+0058 LATIN CAPITAL LETTER X) then let X_1 be "_x0058_".
4. If C_i is not a valid XML NCName character (see [Namespaces in XML](#)) or if $i=1$ (one) and C_1 is not a valid first character of an XML NCName then:

Let U_1, U_2, \dots, U_6 be the six hex digits [PROD: 5] such that C_i is "U+" $U_1 U_2 \dots U_6$ in the Unicode scalar value.

Case:

1. If $U_1=0, U_2=0, U_3=0,$ and $U_4=0,$ then let $X_i="_x" U_5 U_6 "_"$.

This case implies that C_i is a character in the Basic Multilingual Plane (Plane 0) of Unicode and can be wholly represented by a single UTF-16 code point sequence U+ U_5U_6 .

2. Otherwise, let X_i be " $_x" U_1 U_2 U_3 U_4 U_5 U_6 "_"$.

5. Otherwise, let X_i be M_i . That is, any character in X that is a valid character in an XML NCName is simply copied.

7. Let *LocalPart* be the character string concatenation of X_1, X_2, \dots, X_N in order from most to least significant.

8. Let XML Name be the QName per Namespaces in XML [Namespaces in XML]

B.2 Examples

```
Hello world -> Hello_x0020_world
Hello_xorld -> Hello_x005F_xorld
Helloworld_ -> Helloworld_

      x -> x
      xml -> _x0078_ml
     -xml -> _x002D_xml
    x-ml -> x-ml

    Ælfred -> Ælfred
    ἄγνωνστος -> ἄγνωνστος
    □□□□ -> _x1709__x1705__x170E__x1708_
    □□□ -> _x13D9__x13DA__x13A5_
```

C. Using W3C XML Schema with SOAP Encoding (Non-Normative)

As noted in [3.1.4 Computing the Type Name Property](#) SOAP graph nodes are labeled with type names, but conforming processors are not required to perform validation of encoded SOAP messages.

These sections describe techniques that can be used when validation with W3C XML schemas is desired for use by SOAP applications. Any errors or faults resulting from such validation are beyond those covered by the normative recommendation; from the perspective of SOAP, such faults are considered to be application-level failures.

C.1 Validating Using the Minimum Schema

Although W3C XML schemas are conventionally exchanged in the form of schema documents (see XML Schema [\[XML Schema Part 1\]](#)), the schema recommendation is built on an abstract definition of schemas, to which all processors need to conform. The schema recommendation provides that all such schemas include definitions for a core set of built-in types, such as integers, dates, and so on (see XML Schema [\[XML Schema Part 1\]](#), [Built-in Simple Type Definition](#)). Thus, it is possible to discuss validation of a SOAP message against such a minimal schema, which is the one that would result from providing no additional definitions or declarations (i.e. no schema document) to a schema processor.

The minimal schema provides that any well formed XML document will validate, except that where an `xsi:type` is provided, the type named must be built-in, and the corresponding element must be valid per that type. Thus, validation of a SOAP 1.2 message using a minimal schema approximates the behavior of the built-in types of SOAP 1.1.

C.2 Validating Using the SOAP Encoding Schema

Validation against the minimal schema (see [C.1 Validating Using the Minimum Schema](#)) will not succeed where encoded graph nodes have multiple inbound edges. This is because elements representing such graph nodes will carry `id` *attribute information items* which are not legal on elements of type "xs:string", "xs:integer" etc. The SOAP Encoding of such graphs MAY be validated against the [SOAP Encoding schema](#). In order for the encoding to validate, edge labels, and hence the [local name] and [namespace name] properties of the *element information items*, need to match those defined in the SOAP Encoding schema. Validation of the encoded graph against the SOAP Encoding schema would result in the type name property of the nodes in the graph being assigned the relevant type name.

C.3 Validating Using More Specific Schemas

It may be that schemas could be constructed to describe the encoding of certain graphs. Validation of the encoded graph against such a schema would result in the type name property of the graph nodes being assigned the relevant type name. Such a schema can also supply default or fixed values for one or more of the `itemType`, `arraySize` or `nodeType` *attribute information items*; the values of such defaulted attributes affect the deserialized graph in the same manner as if the attributes had been explicitly supplied in the message. Errors or inconsistencies thus introduced (e.g. if the value of the attribute is erroneous or inappropriate) should be reported as application-level errors; faults from the

"<http://www.w3.org/2003/05/soap-encoding>" namespace should be reported only if the normative parts of this specification are violated.

D. Acknowledgements (Non-Normative)

This document 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): Carine Bournez (W3C), Michael Champion (Software AG), Glen Daniels (Macromedia, formerly of Allaire), David Fallside (IBM), Dietmar Gaertner (Software AG), Tony Graham (Sun Microsystems), Martin Gudgin (Microsoft Corporation, formerly of DevelopMentor), Marc Hadley (Sun Microsystems), Gerd Hoelzing (SAP AG), Oisin Hurley (IONA Technologies), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric), Kazunori Iwasa (Fujitsu Limited), Mario Jeckle (DaimlerChrysler R. & Tech), Mark Jones (AT&T), Anish Karmarkar (Oracle), Jacek Kopecky (Systinet/Idoox), Yves Lafon (W3C), Michah Lerner (AT&T), Noah Mendelsohn (IBM, formerly of Lotus Development), Jeff Mischkinsky (Oracle), Nilo Mitra (Ericsson), Jean-Jacques Moreau (Canon), Masahiko Narita (Fujitsu Limited), Eric Newcomer (IONA Technologies), Mark Nottingham (BEA Systems, formerly of Akamai Technologies), David Orchard (BEA Systems, formerly of Jamcracker), Andreas Riegg (DaimlerChrysler R. & Tech), Hervé Ruellan (Canon), Jeff Schlimmer (Microsoft Corporation), Miroslav Simek (Systinet/Idoox), Pete Wenzel (SeeBeyond), Volker Wiechers (SAP AG).

Previous participants were: Yasser alSafadi (Philips Research), Bill Anderson (Xerox), Vidur Apparao (Netscape), Camilo Arbelaez (WebMethods), Mark Baker (Idokorro Mobile (Planetfred), formerly of Sun Microsystems), Philippe Bedu (EDF (Electricité de France)), Olivier Boudeville (EDF (Electricité de France)), Don Box (Microsoft Corporation, formerly of DevelopMentor), Tom Breuel (Xerox), Dick Brooks (Group 8760), Winston Bumpus (Novell), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Cefonkus (Bowstreet), David Chappell (Sonic Software), Miles Chaston (Epicentric), David Clay (Oracle), David Cleary (Progress Software), Conleth O'Connell (Vignette), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Fransisco Cubera (IBM), Jim d'Augustine (eXcelon), Ron Daniel (Interwoven), Dug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE), Frank DeRose (Tibco), Mike Dierken (DataChannel), Andrew Eisenberg (Progress Software), Brian Eisenberg (DataChannel), Colleen Evans (Sonic Software), John Evdemon (XMLSolutions), David Ezell (Hewlett-Packard), Eric Fedok (Active Data Exchange), Chris Ferris (Sun Microsystems), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Michael Freeman (Engenia Software), Scott Golubock (Epicentric), Rich Greenfield (Library of Congress), Hugo Haas (W3C), Mark Hale (Interwoven), Randy Hall (Intel), Bjoern Heckel (Epicentric), Erin Hoffman (Tradia), Steve Hole (MessagingDirect Ltd.), Mary Holstege (Calico Commerce), Jim Hughes (Fujitsu Software Corporation), Yin-Leng Husband (Hewlett-Packard, formerly of Compaq), Scott Isaacson (Novell), Murali Janakiraman (Rogue Wave), Eric Jenkins (Engenia Software), Jay Kasi (Commerce One), Jeffrey Kay (Engenia Software), Richard Koo (Vitria Technology Inc.), Alan Kropp (Epicentric), Julian Kumar (Epicentric), Peter Lecuyer (Progress Software), Tony Lee (Vitria Technology Inc.), Amy Lewis (TIBCO), Bob Lojek (Intalio), Henry Lowe (OMG), Brad Lund (Intel), Matthew MacKenzie

(XMLGlobal Technologies), Murray Maloney (Commerce One), Richard Martin (Active Data Exchange), Highland Mary Mountain (Intel), Alex Milowski (Lexica), Kevin Mitchell (XMLSolutions), Ed Mooney (Sun Microsystems), Dean Moses (Epicentric), Don Mullen (Tibco), Rekha Nagarajan (Calico Commerce), Raj Nair (Cisco), Mark Needleman (Data Research Associates), Art Nevarez (Novell), Henrik Nielsen (Microsoft Corporation), Kevin Perkins (Compaq), Jags Ramnaryan (BEA Systems), Vilhelm Rosenqvist (NCR), Marwan Sabbouh (MITRE), Waqar Sadiq (Vitria Technology Inc.), Rich Salz (Zolera), Krishna Sankar (Cisco), George Scott (Tradia), Shane Sesta (Active Data Exchange), Lew Shannon (NCR), John-Paul Sicotte (MessagingDirect Ltd.), Simeon Simeonov (Allaire), Simeon Simeonov (Macromedia), Aaron Skonnard (DevelopMentor), Nick Smlonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Jim Trezzo (Oracle), Asir Vedamuthu (WebMethods), Randy Waldrop (WebMethods), Fred Waskiewicz (OMG), David Webber (XMLGlobal Technologies), Ray Whitmer (Netscape), Stuart Williams (Hewlett-Packard), Yan Xu (DataChannel), Amr Yassin (Philips Research), Susan Yee (Active Data Exchange), Jin Yu (Martsoft).

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