Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts

W3C Candidate Recommendation 6 January 2006

This version:
http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060106
Latest version:
http://www.w3.org/TR/wsdl20-adjuncts
Previous versions:
http://www.w3.org/TR/2005/WD-wsdl20-adjuncts-20050803
Editors:
Roberto Chinnici, Sun Microsystems
Hugo Haas, W3C
Amelia A. Lewis, TIBCO Software
Jean-Jacques Moreau, Canon
David Orchard, BEA Systems
Sanjiva Weerawarana, WSO2

This document is also available in these non-normative formats: PDF, PostScript, XML, and plain text.

Abstract

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts defines predefined extensions for use in WSDL 2.0:

- Message exchange patterns
- Operation styles
- Binding Extensions

This specification depends on WSDL Version 2.0 [*WSDL 2.0 Core Language [p.70]*].

Status of this Document

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

This is the W3C Candidate Recommendation of Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts for review by W3C Members and other interested parties. It has been produced by the Web Services Description Working Group, which is part of the W3C Web Services Activity. The publication of this document signifies a call for implementations of this specification. This specification will remain a Candidate Recommendation at least until 15 March 2006.

This Working Draft addresses all the comments received during the second Last Call review period on the WSDL 2.0 drafts. The detailed disposition of the comments received can be found in the Last Call issues list. A diff-marked version against the previous version of this document is available. For a detailed list of changes since the last publication of this document, please refer to appendix **C. Part 2 Change Log** [p.74] .

The Working Group plans to submit this specification for consideration as a W3C Proposed Recommendation if the following exit criteria have been met:

- Two interoperable implementations of all the features, both mandatory and optional, of the specifications have been produced.
- The Working Group releases a test suite along with an implementation report.

The following features defined in this specification are considered at risk:

- Serialization of the instance data in parts of the HTTP request IRI (section **6.8.1 Serialization of the instance data in parts of the HTTP request IRI** [p.57]): feedback is sought on this topic
- Definition of the Robust In-Only, In-Optional-Out, Out-Only, Robust Out-Only, Out-In, Out-Optional-In message exchange pattern (in section **2.3 Message Exchange Patterns** [p.11]): the Working Group is intending to remove those definitions from the specification if it does not have evidence of their use

Implementers are encouraged to provide feedback by 15 March 2006. Comments on this document are to be sent to the public public-ws-desc-comments@w3.org mailing list (public archive).

Issues about this document are recorded in the Candidate Recommendation issues list maintained by the Working Group. A list of formal objections against the set of WSDL 2.0 Working Drafts is also available.

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Short Table of Contents

Table of Contents

Appendices

---

1. Introduction

The Web Services Description Language WSDL Version 2.0 (WSDL) [*WSDL 2.0 Core Language [p.70]*] defines an XML language for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication. This document defines extensions for the WSDL 2.0 language:

- Message exchange patterns: **2. Predefined Message Exchange Patterns** [p.9]
- Operation safety declaration: **3. Predefined Extensions** [p.15]
- Operation styles: **4. Predefined Operation Styles** [p.16]
- Binding extensions:
    - A SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework [p.69]*] binding extension: **5. WSDL SOAP Binding Extension** [p.22]
    - An HTTP/1.1 [*IETF RFC 2616 [p.68]*] binding extension: **6. WSDL HTTP Binding Extension** [p.41]

WSDL 2.0 Primer [*WSDL 2.0 Primer [p.70]*] is a non-normative document intended to provide an easily understandable tutorial on the features of the WSDL Version 2.0 specifications.

The Core Language [*WSDL 2.0 Core Language [p.70]*] of the WSDL 2.0 specification describes the core elements of the WSDL language.

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 RFC2119 [*IETF RFC 2119 [p.68]*].

This specification uses a number of namespace prefixes throughout; they are listed in Table 1-1 [p.7] . Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [*XML Information Set [p.70]*]).

Table 1-1. Prefixes and Namespaces used in this specification

Prefix	Namespace	Notes
wsdl	"http://www.w3.org/2006/01/wsdl"	This namespace is defined in [*WSDL 2.0 Core Language [p.70]*]. A normative XML Schema [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*] document for the "http://www.w3.org/2006/01/wsdl" namespace can be found at http://www.w3.org/2006/01/wsdl. This namespace is used as the default namespace throughout this specification.
wsdlx	"http://www.w3.org/2006/01/wsdl-extensions"	This specification extends in section **3. Predefined Extensions** [p.15] the "http://www.w3.org/2006/01/wsdl-extensions" namespace defined in [*WSDL 2.0 Core Language [p.70]*]. A normative XML Schema [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*] document for the "http://www.w3.org/2006/01/wsdl-extensions" namespace can be found at http://www.w3.org/2006/01/wsdl-extensions.
wsoap	"http://www.w3.org/2006/01/wsdl/soap"	Defined by this specification. A normative XML Schema [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*] document for the "http://www.w3.org/2006/01/wsdl/soap" namespace can be found at http://www.w3.org/2006/01/wsdl/soap.
whttp	"http://www.w3.org/2006/01/wsdl/http"	Defined by this specification. A normative XML Schema [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*] document for the "http://www.w3.org/2006/01/wsdl/http" namespace can be found at http://www.w3.org/2006/01/wsdl/http.
wrpc	"http://www.w3.org/2006/01/wsdl/rpc"	Defined by this specification. A normative XML Schema [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*] document for the "http://www.w3.org/2006/01/wsdl/rpc" namespace can be found at http://www.w3.org/2006/01/wsdl/rpc.
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [*XML Schema Structures [p.70]*], [*XML Schema Datatypes [p.70]*].

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [*IETF RFC 3986 [p.69]*].

All parts of this specification are normative, with the EXCEPTION of pseudo-schemas, examples, and sections explicitly marked as "Non-Normative". Pseudo-schemas are provided for each component, before the description of this component. They provide visual help for the XML [*XML 1.0 [p.70]*] serialization. The syntax of BNF pseudo-schemas is the same as the one used in [*WSDL 2.0 Core Language [p.70]*].

2. Predefined Message Exchange Patterns

Web Services Description Language (WSDL) message exchange patterns (hereafter simply 'patterns') define the sequence and cardinality of abstract messages listed in an operation. Message exchange patterns also define which other nodes send messages to, and receive messages from, the service implementing the operation.

A *node* is an agent (section 2.3.2.2 Agent of the Web Services Architecture [*Web Services Architecture [p.69]*]) that can transmit and/or receive message(s) described in WSDL description(s) and process them.

Note:

A node may be accessible via more than one physical address or transport.

WSDL message exchange patterns describe the interaction at the abstract (interface) level, which may be distinct from the pattern used by the underlying protocol binding (e.g. SOAP Message Exchange Patterns; section **5.10.3 SOAP 1.2 Binding Rules** [p.38] contains the binding rules for the selection of a SOAP 1.2 message exchange pattern based on the WSDL message exchange pattern in use for the SOAP binding extension defined in this specification in section **5. WSDL SOAP Binding Extension** [p.22]).

By design, WSDL message exchange patterns abstract out specific message types. Patterns identify placeholders for messages, and placeholders are associated with specific message types by the operation using the pattern.

Unless explicitly stated otherwise, WSDL message exchange patterns also abstract out binding-specific information like timing between messages, whether the pattern is synchronous or asynchronous, and whether the message are sent over a single or multiple channels.

Like interfaces and operations, WSDL message exchange patterns do not exhaustively describe the set of messages exchanged between a service and other nodes; by some prior agreement, another node and/or the service may send other messages (to each other or to other nodes) that are not described by the pattern. For instance, even though a pattern may define a single message sent from a service to one other node, the Web Service may multicast that message to other nodes.

To maximize reuse, WSDL message exchange patterns identify a minimal contract between other parties and Web Services, and contain only information that is relevant to both the Web Service and another party.

This specification defines several message exchange patterns for use with *WSDL Version 2.0 Part 1: Core Language* [*WSDL 2.0 Core Language [p.70]*].

2.1 Template for Message Exchange Patterns

New Message Exchange Patterns may be defined by any organization able and willing to do so. It is recommended that the patterns use the general template provided here, after examination of existing predefined patterns.

2.1.1 Pattern Name

This pattern consists of [number] message[s, in order] as follows:

[enumeration, specifying, for each message] A[n optional] message:

1. indicated by a Interface Message Reference component whose {message label} is "[label]" and {direction} is "[direction]"
2. [received from|sent to] ['some' if first mention] node [node identifier]

This pattern uses the rule [fault ruleset reference].

An operation using this message exchange pattern has a {message exchange pattern} property with the value "[pattern IRI]".

Note: In the template, the bracketed items indicate a replacement operation. Substitute the correct terms for each bracketed item.

Note: the "received from" and "sent to" are always from the point of view of the service, and participating nodes other than the service are implicitly identified as the originators of or destinations for messages in the exchange.

2.2 Fault Propagation Rules

WSDL patterns specify their fault propagation model using standard rulesets to indicate where faults may occur. The most common patterns for fault propagation are defined here, and referenced by patterns later in the document. "Propagation" is defined as a best-effort attempt to transmit the fault message to its designated recipient.

WSDL patterns specify propagation of faults, not their generation. Nodes which generate a fault MUST attempt to propagate the faults in accordance with the governing ruleset, but it is understood that any delivery of a network message is best effort, not guaranteed. The rulesets establish the direction of the fault message and the fault recipient, they do not provide reliability or other delivery guarantees. When a fault is generated, the generating node MUST attempt to propagate the fault, and MUST do so in the direction and to the recipient specified by the ruleset. However, extensions or binding extensions MAY modify these rulesets. For example, WS-Addressing [*WSA 1.0 Core [p.70]*] defines a "FaultTo" address for messages, which is used in lieu of the recipient nominated by the ruleset.

Generation of a fault, regardless of ruleset, terminates the exchange.

Binding extensions, features, or extension specifications may override the semantics of a fault propagation ruleset, but this practice is strongly discouraged.

2.2.1 Fault Replaces Message

Any message after the first in the pattern MAY be replaced with a fault message, which MUST have identical direction. The fault message MUST be delivered to the same target node as the message it replaces, unless otherwise specified by an extension or binding extension. If there is no path to this node, the fault MUST be discarded.

This fault propagation rule is identified by the following URI:
`http://www.w3.org/2006/01/wsdl/fault-replaces-message`

2.2.2 Message Triggers Fault

Any message, including the first in the pattern, MAY trigger a fault message, which MUST have opposite direction. The fault message MUST be delivered to the originator of the triggering message, unless otherwise specified by an extension of binding extension. Any node MAY propagate a fault message, and MUST not do so more than once for each triggering message. If there is no path to the originator, the fault MUST be discarded.

This fault propagation rule is identified by the following URI:
`http://www.w3.org/2006/01/wsdl/message-triggers-fault`

2.2.3 No Faults

Faults MUST NOT be propagated.

This fault propagation rule is identified by the following URI:
`http://www.w3.org/2006/01/wsdl/no-faults`

2.3 Message Exchange Patterns

WSDL patterns are described in terms of the WSDL component model, specifically the Interface Message Reference and Interface Fault Reference components.

2.3.1 In-Only

This pattern consists of exactly one message as follows:

1. A message:

    - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"

- received from some node N

This pattern uses the rule **2.2.3 No Faults** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/in-only".

2.3.2 Robust In-Only

This pattern consists of exactly one message as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"
   - received from some node N

This pattern uses the rule **2.2.2 Message Triggers Fault** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/robust-in-only".

2.3.3 In-Out

This pattern consists of exactly two messages, in order, as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"
   - received from some node N
2. A message:
   - indicated by a Interface Message Reference component whose {message label} is "Out" and {direction} is "out"
   - sent to node N

This pattern uses the rule **2.2.1 Fault Replaces Message** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/in-out".

2.3.4 In-Optional-Out

This pattern consists of one or two messages, in order, as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"
   - received from some node N
2. An optional message:
   - indicated by a Interface Message Reference component whose {message label} is "Out" and {direction} is "out"
   - sent to node N

This pattern uses the rule **2.2.2 Message Triggers Fault** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/in-opt-out".

2.3.5 Out-Only

This pattern consists of exactly one message as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "Out " and {direction} is "out"
   - sent to some node N

This pattern uses the rule **2.2.3 No Faults** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/out-only".

2.3.6 Robust Out-Only

This pattern consists of exactly one message as follows:

1. message:
   - indicated by a Interface Message Reference component whose {message label} is "Out" and {direction} is "out"

- sent to some node N

This pattern uses the rule **2.2.2 Message Triggers Fault** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/robust-out-only".

2.3.7 Out-In

This pattern consists of exactly two messages, in order, as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "Out" and {direction} is "out"
   - sent to some node N
2. A message:
   - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"
   - sent from node N

This pattern uses the rule **2.2.1 Fault Replaces Message** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/out-in".

2.3.8 Out-Optional-In

This pattern consists of one or two messages, in order, as follows:

1. A message:
   - indicated by a Interface Message Reference component whose {message label} is "Out" and {direction} is "out"
   - sent to some node N
2. An optional message:
   - indicated by a Interface Message Reference component whose {message label} is "In" and {direction} is "in"
   - sent from node N

This pattern uses the rule **2.2.2 Message Triggers Fault** [p.11] .

An operation using this message exchange pattern has a {message exchange pattern} property with the value "http://www.w3.org/2006/01/wsdl/out-opt-in".

2.4 Security Considerations

Note that many of the message exchange patterns defined above describe responses to an initial message (either a normal response message or a fault.)

Such responses may be used in attempts to disrupt, attack, or map a network, host, or services. When such responses are directed to an address other than that originating the initial message, the source of an attack may be obscured, or blame laid on a third party, or may enable or exacerbate denial-of-service attacks.

Security mechanisms addressing such attacks may prevent the delivery of response messages to the receiving node. Conformance to the message exchange pattern is measured prior to the application of these security mechanisms.

3. Predefined Extensions

3.1 Operation safety

This section defines an extension to WSDL 2.0 [*WSDL 2.0 Core Language [p.70]*] which allows to mark an operation as a safe interaction, as defined in section 3.4. Safe Interactions of [*Web Architecture [p.69]*].

This extension MAY be used for setting defaults in bindings, such as in an HTTP binding per this specification (see **6.5.5 Mapping from XML Representation to Component Properties** [p.50]).

3.1.1 Relationship to WSDL Component Model

The safety extension adds the following property to the Interface Operation component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {safety} REQUIRED. An *xs:boolean* indicating whether the operation is asserted to be safe for users of the described service to invoke. If this property is "false", then no assertion has been made about the safety of the operation, thus the operation MAY or MAY NOT be safe. However, an operation SHOULD be marked safe if it meets the criteria for a safe interaction defined in Section 3.5 of [*Web Architecture [p.69]*].

3.1.2 XML Representation

```
<description>
 <interface>
   <operation name="xs:NCName" pattern="xs:anyURI"
```

```
               wsdlx:safe="xs:boolean"? >
  </operation>
 </interface>
</description>
```

The XML representation for the safety extension is an *attribute information item* with the following Infoset properties:

- An OPTIONAL `safe` *attribute information item* with the following Infoset properties:
  - A [local name] of `safe`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl-extensions"
  - A type of *xs:boolean*

3.1.3 Mapping from XML Representation to Component Properties

See Table 3-1 [p.16] .

Table 3-1. Mapping from XML Representation to Interface Operation component Extension Properties

Property	Value
{safety [p.15] }	The actual value of the `safe` *attribute information item*, if present, otherwise the value "false".

4. Predefined Operation Styles

This section defines operation styles that can be used to place constraints on Interface Operation components, in particular with respect to the format of the messages they refer to. The serialization formats defined in section **6.8 Serialization Format of Instance Data** [p.55] require bound Interface Operation components to have one or more of the styles defined in this section.

4.1 RPC Style

The RPC style is selected by assigning to an Interface Operation component's {style} property the value "http://www.w3.org/2006/01/wsdl/style/rpc".

In order to conform with the specification for the RPC style, an Interface Operation component MUST obey the constraints listed below. Furthermore, if the `wrpc:signature` extension is used, the corresponding *attribute information item* MUST be valid according to the schema for the extension and additionally MUST obey the constraints listed in **4.1.1 wrpc:signature Extension** [p.17] and **4.1.2 XML Representation of the wrpc:signature Extension** [p.19] .

The RPC style MUST NOT be used for Interface Operation components whose {message exchange pattern} property has a value other than "http://www.w3.org/2006/01/wsdl/in-only" or "http://www.w3.org/2006/01/wsdl/in-out".

The RPC style places restrictions for Remote Procedure Call-types of interactions. When this value is used, the associated messages MUST conform to the rules below, described using XML Schema [*XML Schema Structures [p.70]*]. Note that operations containing messages described by other type systems may also indicate use of the RPC style, as long as they are constructed in such a way as to follow these rules.

If the Interface Operation component uses a {message exchange pattern} for which there is no output element, i.e. "http://www.w3.org/2006/01/wsdl/in-only", then the conditions stated below that refer to output elements MUST be considered to be implicitly satisfied.

- The value of the {message content model} property for the Interface Message Reference components of the {interface message references} property MUST be "#element".
- The content model of input and output {element declaration} elements MUST be defined using a complex type that contains a sequence from XML Schema.
- The input sequence MUST only contain elements and element wildcards. It MUST NOT contain other structures such as xs:choice. The input sequence MUST NOT contain more than one element wildcard. The element wildcard, if present, MUST appear after any elements.
- The output sequence MUST only contain elements. It MUST NOT contain other structures such as xs:choice.
- The sequence MUST contain only local element children. Note that these child elements MAY contain the following attributes: nillable, minOccurs and maxOccurs.
- The local name of input element's QName MUST be the same as the Interface Operation component's name.
- Input and output elements MUST both be in the same namespace.
- The complex type that defines the body of an input or an output element MUST NOT contain any local attributes. Extension attributes are allowed for purposes of managing the message infrastructure (e.g. adding identifiers to facilitate digitally signing the contents of the message). They must not be considered as part of the application data that is conveyed by the message. Therefore, they are never included in an RPC signature (see **4.1.1 wrpc:signature Extension** [p.17]).
- If elements with the same qualified name appear as children of both the input and output elements, then they MUST both be declared using the same named type.
- The input or output sequence MUST NOT contain multiple children elements declared with the same name.

4.1.1 `wrpc:signature` Extension

The `wrpc:signature` extension *attribute information item* MAY be used in conjunction with the RPC style to describe the exact signature of the function represented by an operation that uses the RPC style.

When present, the `wrpc:signature` extension contributes the following property to the Interface Operation component it is applied to:

- {rpc signature} REQUIRED. A list of pairs *(q, t)* whose first component is of type *xs:QName* and whose second component is of type *xs:token*. Values for the second component MUST be chosen among the following four: "#in", "#out", "#inout" "#return".

The value of the {rpc signature [p.18] } property MUST satisfy the following conditions:

- The value of the first component of each pair *(q, t)* MUST be unique within the list.
- For each child element of the input and output messages of the operation, a pair *(q, t)* whose first component *q* is equal to the qualified name of that element MUST be present in the list, with the caveat that elements that appear with cardinality greater than one MUST be treated as a single element.
- For each pair *(q, #in)*, there MUST be a child element of the input element with a name of *q* and there MUST NOT be a child element of the output element with the same name.
- For each pair *(q, #out)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.
- For each pair *(q, #inout)*, there MUST be a child element of the input element with a name of *q* and there MUST be a child element of the output element with the same name. Furthermore, those two elements MUST have the same type.
- For each pair *(q, #return)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.

The function signature defined by a `wrpc:signature` extension is determined as follows:

1. Start with the value of the {rpc signature [p.18] } property, a (possibly empty) list of pairs of this form:

   [(q0, t0), (q1, t1), ...]

2. Filter the elements of this list into two lists, the first one *(L1)* comprising pairs whose *t* component is one of *{#in, #out, #inout}*, the second *(L2)* pairs whose *t* component is *#return*. During the composition of *L1* and *L2*, the relative order of members in the original list MUST be preserved.

   For ease of visualization, let’s denote the two lists as

   (L1) *[(a0, u0), (a1, u1),...]*

   and

   (L2) *[(r0, #return), (r1, #return),...]*

respectively.

3. Then, if the input sequence ends with an element wildcard, the formal signature of the function is

   f([d0] a0, [d1] a1, ..., rest) => (r0, r1, ...)

   where *rest* is a formal parameter representing the elements in the input message matched by the element wildcard.

   Otherwise the formal signature of the function is

   f([d0] a0, [d1] a1, ...) => (r0, r1, ...)

   i.e.

   - the list of formal arguments to the function is *[a0, a1, ...]*;
   - the direction *d* of each formal argument *a* is one of *[in]*, *[out]*, *[inout]*, determined according to the value of its corresponding *u* token;
   - the list of formal return parameters of the function is *[r0, r1, ...]*;
   - each formal argument and formal return parameter is typed according to the type of the child element identified by it (unique per the conditions given above).

Note:

The `wrpc:signature` extension allows the specification of multiple return values for an operation. Several popular programming languages support multiple return values for a function. Moreover, for languages which do not, the burden on implementors should be small, as typically multiple return values will be mapped to a single return value of a structure type (or its closest language-specific equivalent).

4.1.2 XML Representation of the `wrpc:signature` Extension

The XML representation for the RPC signature extension is an *attribute information item* with the following Infoset properties:

- A [local name] of `signature`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/rpc"

The type of the `name` *attribute information item* is a list type whose item type is the union of the *xs:QName* type and the subtype of the *xs:token* type restricted to the following four values: "#in", "#out", "#inout", "#return". See Example 4-1 [p.20] for an excerpt from the normative schema definition of this type.

Additionally, each even-numbered item (0, 2, 4, ...) in the list MUST be of type *xs:QName* and each odd-numbered item (1, 3, 5, ...) in the list MUST be of the subtype of *xs:token* described in the previous paragraph.

Example 4-1. Definition of the wrpc:signature extension

```
<xs:attribute name="signature" type="wrpc:signatureType"/>

<xs:simpleType name="signatureType">
  <xs:list itemType="wrpc:signatureItemType"/>
</xs:simpleType>

<xs:simpleType name="signatureItemType">
  <xs:union memberTypes="xs:QName wrpc:directionToken"/>
</xs:simpleType>

<xs:simpleType name="directionToken">
  <xs:restriction base="xs:token">
    <xs:enumeration value="#in"/>
    <xs:enumeration value="#out"/>
    <xs:enumeration value="#inout"/>
    <xs:enumeration value="#return"/>
  </xs:restriction>
</xs:simpleType>
```

4.1.3 `wrpc:signature` Extension Mapping To Properties of an Interface Operation component

A `wrpc:signature` extension *attribute information item* is mapped to the following property of the Interface Operation component defined by its [owner].

Table 4-1. Mapping of a `wrpc:signature` Extension to Interface Operation component Properties

Property	Value
{rpc signature [p.18] }	A list of *(xs:QName, xs:token)* pairs formed by grouping the items present in the actual value of the `wrpc:signature` *attribute information item* in the order in which they appear there.

4.2 IRI Style

The IRI style is selected by assigning the Interface Operation component's {style} property the value "http://www.w3.org/2006/01/wsdl/style/iri".

When using this style, the value of the {message content model} property of the Interface Message Reference component corresponding to the initial message of the message exchange pattern MUST be "#element".

Use of this value indicates that XML Schema [*XML Schema Structures [p.70]*] was used to define the schema of the {element declaration} property of the Interface Message Reference component of the Interface Operation component corresponding to the initial message of the message exchange pattern. This schema MUST adhere to the rules below:

- The content model of this element is defined using a complex type that contains a sequence from XML Schema.
- The sequence MUST only contain elements. It MUST NOT contain other structures such as xs:choice.
- The sequence MUST contain only local element children. These child elements MAY contain the `nillable` attribute, and the attributes `minOccurs` and `maxOccurs` MUST have a value `0` or `1`.
- The localPart of the element's QName MUST be the same as the Interface Operation component's {name}.
- The complex type that defines the body of the element or its children elements MUST NOT contain any attributes.
- The sequence MUST NOT contain multiple children elements declared with the same local name.
- If the children elements of the sequence are defined using an XML Schema type, they MUST derive from `xs:simpleType`, and MUST NOT be of the type or derive from `xs:QName`, `xs:NOTATION`, `xs:hexBinary` or `xs:base64Binary`.

4.3 Multipart style

The Multipart style is selected by assigning the Interface Operation component's {style} property the value "http://www.w3.org/2006/01/wsdl/style/multipart".

When using this style, the value of the {message content model} property of the Interface Message Reference component corresponding to the initial message of the message exchange pattern MUST be "#element".

Use of this value indicates that XML Schema [*XML Schema Structures [p.70]*] was used to define the schema of the {element declaration} property of the Interface Message Reference component of the Interface Operation component corresponding to the initial message of the message exchange pattern. This schema MUST adhere to the rules below:

- The content model of this element is defined using a complex type that contains a sequence from XML Schema.
- The sequence MUST only contain elements. It MUST NOT contain other structures such as xs:choice.
- The sequence MUST contain only local element children. These child elements MAY contain the `nillable` attribute, and the attributes `minOccurs` and `maxOccurs` MUST have a value `1`.
- The localPart of the element's QName MUST be the same as the Interface Operation component's {name}.

- The complex type that defines the body of the element or its children elements MUST NOT contain any attributes.
- The sequence MUST NOT contain multiple children element declared with the same local name.

5. WSDL SOAP Binding Extension

The SOAP binding extension described in this section is SOAP version independent ("1.2" as well as other versions) and an extension for [*WSDL 2.0 Core Language [p.70]*] to enable Web Services applications to use SOAP. This binding extension extends WSDL 2.0 by adding properties to the Binding component as defined in [*WSDL 2.0 Core Language [p.70]*]. In addition, an XML Infoset representation for these additional properties is provided, along with a mapping from that representation to the various component properties.

As allowed in [*WSDL 2.0 Core Language [p.70]*], a Binding component MAY exist without indicating a specific Interface component that it applies to. In this case, there MUST NOT be any Binding Operation or Binding Fault components present in the Binding component.

The SOAP binding extension is designed with the objective of minimizing what needs to be explicitly declared for common cases. This is achieved by defining a set of default rules which apply for all Interface Operation components of an Interface component, unless specifically overridden on a per Interface Operation basis. Thus, if a given Interface Operation component is not referred to specifically, then all the default rules apply for that component. That is, per the requirements of [*WSDL 2.0 Core Language [p.70]*], all operations of an Interface component are bound according to this binding extension.

A subset of the HTTP properties specified in the HTTP binding extension defined in section **6. WSDL HTTP Binding Extension** [p.41] may be expressed in a SOAP binding when the SOAP binding uses HTTP as the underlying protocol, for example, when the value of the {soap underlying protocol [p.27] } property of the Binding component is "http://www.w3.org/2003/05/soap/bindings/HTTP/". The properties that are allowed are the ones that describe the underlying protocol:

- {http version [p.46] } on Binding components, as defined in **6.4 Specifying the HTTP Version** [p.46]
- {http location [p.47] } on Binding Operation components, as defined in **6.5 Binding Operations** [p.47]
- {http headers [p.51] } on Binding Message Reference and Binding Fault components, as defined in **6.6 Declaring HTTP Headers** [p.51]
- {http transfer coding default [p.64] } on Binding and Binding Operation components, {http transfer coding [p.64] } on Binding Message Reference and Binding Fault components, as defined in **6.9 Specifying the Transfer Coding** [p.63]
- {http cookies [p.66] } on Binding components, as defined in **6.10 Specifying the Use of HTTP Cookies** [p.65]

- {http authentication scheme [p.67] } and {http authentication realm [p.67] } on Endpoint components, as defined in **6.11 Specifying HTTP Access Authentication** [p.66]

5.1 XML Syntax Summary (Non-Normative)

```
<description>
  <binding name="xs:NCName" interface="xs:QName"?
           type="http://www.w3.org/2006/01/wsdl/soap"
           whttp:version="xs:string"??
           whttp:transferCodingDefault="xs:string"??
           wsoap:version="xs:string"?
           wsoap:protocol="xs:anyURI"
           wsoap:mepDefault="xs:anyURI"? >
    <documentation />*

    <wsoap:module ref="xs:anyURI" required="xs:boolean"? >
      <documentation />*
    </wsoap:module>*

    <fault ref="xs:QName"
           wsoap:code="union of xs:QName, xs:token"?
           wsoap:subcodes="list of xs:QName"?
           whttp:transferCoding="xs:string"?? >

      <documentation />*

      <wsoap:module ... />*
      <wsoap:header element="xs:QName" mustUnderstand="xs:boolean"?
                    required="xs:boolean"? >
        <documentation />*
      </wsoap:header>*
      <whttp:header ... />*??

      [ <feature /> | <property /> ]*
    </fault>*

    <operation ref="xs:QName"
               whttp:location="xs:anyURI"??
               whttp:transferCodingDefault="xs:string"?? >
               wsoap:mep="xs:anyURI"?
               wsoap:action="xs:anyURI"? >

      <documentation />*

      <wsoap:module ... />*

      <input messageLabel="xs:NCName"?
             whttp:transferCoding="xs:string"?? >
        <documentation />*
        <wsoap:module ... />*
        <wsoap:header ... />*
        <whttp:header ... />*??
        [ <feature /> | <property /> ]*
      </input>*

      <output messageLabel="xs:NCName"?
```

```
             whttp:transferCoding="xs:string"?? >
        <documentation />*
        <wsoap:module ... />*
        <wsoap:header ... />*
        <whttp:header ... />*??
        [ <feature /> | <property /> ]*
      </output>*

      <infault ref="xs:QName"
                 messageLabel="xs:NCName"?>
        <documentation />*
        <wsoap:module ... />*
        [ <feature /> | <property /> ]*
      </infault>*

      <outfault ref="xs:QName"
                  messageLabel="xs:NCName"?>
        <documentation />*
        <wsoap:module ... />*
        [ <feature /> | <property /> ]*
      </outfault>*

      [ <feature /> | <property /> ]*

    </operation>*

    [ <feature /> | <property /> ]*

  </binding>

  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"?
              whttp:authenticationType="xs:token"??
              whttp:authenticationRealm="xs:string"?? >
      <documentation />*
      [ <feature /> | <property /> ]*
    </endpoint>
    [ <feature /> | <property /> ]*
  </service>
</description>
```

Note:

The double question marks ("`??`") after the attributes in the `whttp` namespace indicates that those optional attributes only make sense when the SOAP binding uses HTTP as the underlying protocol, for example, when the value of the `wsoap:protocol` attribute is "http://www.w3.org/2003/05/soap/bindings/HTTP/".

5.2 Identifying the use of the SOAP Binding

A Binding component (defined in [*WSDL 2.0 Core Language [p.70]*]) is identified as a SOAP binding by assigning the value "http://www.w3.org/2006/01/wsdl/soap" to the {type} property of the Binding component.

5.3 SOAP Binding Rules

- *Payload Construction.* When formulating the SOAP envelope to be transmitted, the contents of the payload (i.e., the contents of the SOAP Body *element information item* of the SOAP envelope) MUST be what is defined by the corresponding Interface Message Reference component. This is subject to optimization by a feature that is in use which may affect serialization, such as MTOM [*SOAP Message Transmission Optimization Mechanism [p.70]*]. The following binding rules MUST be adhered to:

  - If the value of the {message content model} property of the Interface Message Reference component is "#any" then the payload MAY be any one XML element.

  - If the value is "#none" then the payload MUST be empty.

  - If the value is "#element" then the payload will be the *element information item* identified by the {element declaration} property of the Interface Message Reference component.

  - If the Interface Message Reference component is declared using a non-XML type system (as considered in the Types section of [*WSDL 2.0 Core Language [p.70]*]) then additional binding rules MUST be defined to indicate how to map those components into the SOAP envelope.

  Note:

  This SOAP binding extension only allows one single element in SOAP body.

- *SOAP Header Construction.* If the {soap headers [p.34] } property as defined in section **5.9 Declaring SOAP Header Blocks** [p.34] exists and is not empty in a Binding Message Reference or Binding Fault component, *element information item* conforming to the element declaration of a SOAP Header Block [p.34] component's {element declaration [p.34] } property, in the {soap headers [p.34] } property, MAY be turned into a SOAP header block for the corresponding message.

  If the value of the SOAP Header Block [p.34] component's {required [p.35] } property is "true", the inclusion of this SOAP header block is REQUIRED, otherwise it is OPTIONAL.

  And, if the SOAP Header Block [p.34] component's {mustUnderstand [p.34] } property is present and its value is "true", that particular SOAP header block should be marked with a `mustUnderstand` *attribute information item* with a value of "true" or "1" as per the SOAP specification.

  SOAP header blocks other than the ones declared in the {soap headers [p.34] } property may be present at run-time, such as the SOAP header blocks resulting from SOAP modules declared as explained in section **5.8 Declaring SOAP Modules** [p.31] .

5.4 Specifying the SOAP Version

5.4.1 Description

Every SOAP binding MUST indicate what version of SOAP is in use for the operations of the interface that this binding applies to.

By default, SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework [p.69]*] is used.

5.4.2 Relationship to WSDL Component Model

The SOAP protocol specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap version} REQUIRED. A *xs:string*, to the Binding component.

5.4.3 XML Representation

```
<description>
  <binding  name="xs:NCName" interface="xs:QName"? type="xs:anyURI"
            wsoap:version="xs:string"? >
    ...
  </binding>
</description>
```

The XML representation for specifying the SOAP version is an optional *attribute information item* with the following Infoset properties:

- A [local name] of `version`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
- A type of *xs:string*

5.4.4 Mapping from XML Representation to Component properties

See Table 5-1 [p.26] .

Table 5-1. Mapping from XML Representation to Binding component Extension Properties

Property	Value
{soap version [p.26] }	The actual value of the `wsoap:version` *attribute information item* if present, otherwise "1.2".

5.5 Specifying the SOAP Underlying Protocol

5.5.1 Description

Every SOAP binding MUST indicate what underlying protocol is in use.

5.5.2 Relationship to WSDL Component Model

The SOAP protocol specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap underlying protocol} REQUIRED. A *xs:anyURI*, which is an absolute IRI as defined by [*IETF RFC 3987 [p.69]*], to the Binding component.

5.5.3 XML Representation

```
<description>
  <binding  name="xs:NCName" interface="xs:QName"? type="xs:anyURI"
            wsoap:protocol="xs:anyURI" >
    ...
  </binding>
</description>
```

The XML representation for specifying the SOAP protocol is a REQUIRED *attribute information item* with the following Infoset properties:

- A [local name] of `protocol`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
- A type of *xs:anyURI*

5.5.4 Mapping from XML Representation to Component Properties

See Table 5-2 [p.27] .

Table 5-2. Mapping from XML Representation to Binding component Extension Properties

Property	Value
{soap underlying protocol [p.27] }	The actual value of the `wsoap:protocol` *attribute information item.*

5.6 Binding Faults

5.6.1 Description

For every Interface Fault component contained in an Interface component, a mapping to a SOAP Fault must be described. This binding extension specification allows the user to indicate the SOAP fault code and subcodes that are transmitted for a given Interface Fault component.

5.6.2 Relationship to WSDL Component Model

The SOAP Fault binding extension adds the following properties to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap fault code} REQUIRED. A union of *xs:QName* and *xs:token* where the allowed token value is "#any", to the Binding Fault component. The value of this property identifies a possible SOAP fault for the operations in scope. If the value of this property is "#any", no assertion is made about the value of the SOAP fault code.
- {soap fault subcodes} REQUIRED. A union of list of *xs:QName*, and *xs:token* where the allowed token value is "#any", to the Binding Fault component. The value of this property identifies one or more subcodes for this SOAP fault. If the value of this property is "#any", no assertion is made about the value of the SOAP fault subcode.

5.6.3 XML Representation

```
<description>
  <binding >
    <fault ref="xs:QName"
           wsoap:code="union of xs:QName, xs:token"?
           wsoap:subcodes="list of xs:QName"? >
      <documentation />*
      [ <feature /> | <property /> ]*
    </fault>*
  </binding>
</description>
```

The XML representation for binding a SOAP Fault are two *attribute information item*s with the following Infoset properties:

- wsoap:code OPTIONAL *attribute information item*
  - A [local name] of `code`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
  - A type of union of *xs:QName* and *xs:token* where the allowed token value is "#any"
- wsoap:subcodes OPTIONAL *attribute information item*
  - A [local name] of `subcodes`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
  - A type of union of list of *xs:QName*, and *xs:token* where the allowed token value is "#any"

5.6.4 Mapping XML Representation to Component Properties

See Table 5-3 [p.29] .

Table 5-3. Mapping from XML Representation to SOAP Fault component Properties

Property	Value
{soap fault code [p.28] }	The actual value of the `code` *attribute information item* if present; otherwise "#any".
{soap fault subcodes [p.28] }	The actual value of the `subcodes` *attribute information item*, if present; otherwise "#any".

5.7 Binding Operations

5.7.1 Description

For every Interface Operation component contained in an Interface component, in addition to the binding rules (for SOAP 1.2, see **5.10.3 SOAP 1.2 Binding Rules** [p.38]), there may be additional binding information to be specified. This binding extension specification allows the user to indicate the SOAP Message Exchange Pattern (MEP) and a value for the SOAP Action Feature on a per-operation basis.

5.7.2 Relationship to WSDL Component Model

The SOAP Operation binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap mep default} OPTIONAL. A *xs:anyURI*, which is an absolute IRI as defined by [*IETF RFC 3987 [p.69]*], to the Binding component. The value of this property identifies the default SOAP Message Exchange Pattern (MEP) for all the Interface Operation components of any Interface component that uses this Binding component.

- {soap mep} OPTIONAL. A *xs:anyURI*, which is an absolute IRI as defined by [*IETF RFC 3987 [p.69]*], to the Binding Operation component. The value of this property identifies the SOAP Message Exchange Pattern (MEP) for this specific operation.

- {soap action} OPTIONAL. A *xs:anyURI*, which is an absolute IRI as defined by [*IETF RFC 3987 [p.69]*], to the Binding Operation component. The value of this property identifies the value of the SOAP Action Feature for the initial message of the message exchange pattern of the Interface Operation bound, as specified in the binding rules of bindings to specific versions of SOAP (see **5.10.3 SOAP 1.2 Binding Rules** [p.38] for the SOAP 1.2 binding when the value of the {soap version [p.26] } property of the Binding component is "1.2").

5.7.3 XML Representation

```
<description>
  <binding wsoap:mepDefault="xs:anyURI"? >
    <operation ref="xs:QName"
               wsoap:mep="xs:anyURI"?
               wsoap:action="xs:anyURI"? >
    </operation>
  </binding>
</description>
```

The XML representation for binding an Operation are two *attribute information item*s with the following Infoset properties:

- wsoap:mep OPTIONAL *attribute information item*
  - A [local name] of `mep`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
  - A type of *xs:anyURI*
- wsoap:action OPTIONAL *attribute information item*
  - A [local name] of `action`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
  - A type of *xs:anyURI*

The following *attribute information item* for the `binding` *element information item* is defined:

- A [local name] of `mepDefault`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
- A type of *xs:anyURI*

5.7.4 Mapping from XML Representation to Component Properties

See Table 5-4 [p.30] .

Table 5-4. Mapping from XML Representation to SOAP Operation Component Properties

Property	Value
{soap mep default [p.29] }	The actual value of the `wsoap:mepDefault` *attribute information item*, if present.
{soap mep [p.29] }	The actual value of the `wsoap:mep` *attribute information item*, if present.

{soap action [p.29] }	The actual value of the `action` *attribute information item*, if any.

5.8 Declaring SOAP Modules

5.8.1 Description

The SOAP messaging framework allows to engage one or more additional features (typically implemented as one or more SOAP header blocks), as defined by SOAP Modules (see [*SOAP 1.2 Part 1: Messaging Framework [p.69]*]). This binding extension specification allows users to indicate which SOAP Modules are in use across an entire binding, on a per operation basis or on a per message basis.

5.8.2 Relationship to WSDL Component Model

The SOAP Module [p.31] component adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap modules} OPTIONAL. A set of SOAP Module [p.31] components as defined in **5.8.3 SOAP Module component** [p.31] to the Binding component
- Similarly, {soap modules} OPTIONAL, to the Binding Operation component
- Similarly, {soap modules} OPTIONAL, to the Binding Message Reference component
- Similarly, {soap modules} OPTIONAL, to the Binding Fault component
- Similarly, {soap modules} OPTIONAL, to the Binding Fault Reference component

The SOAP modules applicable for a particular operation of any service consists of all modules specified in the input or output Binding Message Reference components, the infault or outfault Binding Fault Reference components, those specified within the Binding Fault components, those specified within the Binding Operation components and those specified within the Binding component. If any module is declared in multiple components, then the requiredness of that module is defined by the closest declaration, where closeness is defined by whether it is specified directly at the Binding Message Reference component or Binding Fault Reference component level, the Binding Fault level or the Binding Operation component level or the Binding component level, respectively.

5.8.3 SOAP Module component

The SOAP Module [p.31] component identifies a SOAP module that is in use.

The properties of the SOAP Module component are as follows:

- {ref} REQUIRED. A *xs:anyURI*, which is an absolute IRI as defined by [*IETF RFC 3987 [p.69]*]. The value of this property identifies the specific SOAP module that is in use.

- {required} REQUIRED. A *xs:boolean* indicating if the SOAP module is required.
- {parent} REQUIRED. The Binding, Binding Operation, Binding Message Reference, Binding Fault or Binding Fault Reference component component that contains this component in its {soap modules [p.31] } property.

5.8.4 XML Representation

```
<description>
  <binding >
    <wsoap:module ref="xs:anyURI"
                  required="xs:boolean"? >
      <documentation ... />*
    </wsoap:module>
    <fault>
      <wsoap:module ... />*
    </fault>
    <operation>
      <wsoap:module ... />*
      <input>
        <wsoap:module ... />*
      </input>
      <output>
        <wsoap:module ... />*
      </output>
      <infault>
        <wsoap:module ... />*
      </infault>
      <outfault>
        <wsoap:module ... />*
      </outfault>
    </operation>
  </binding>
</description>
```

The XML representation for a SOAP Module [p.31] component is an *element information item* with the following Infoset properties:

- A [local name] of `module`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* with the following Infoset properties:
    - A [local name] of `ref`
    - A [namespace name] which has no value
    - A type of *xs:anyURI*

  - An OPTIONAL `required` *attribute information item* with the following Infoset properties:
    - A [local name] of `required`
    - A [namespace name] which has no value
    - A type of *xs:boolean*
  - Zero or more namespace qualified *attribute information items*. The [namespace name] of such *attribute information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/soap".

- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* as defined in [*WSDL 2.0 Core Language [p.70]*].
  2. Zero or more namespace-qualified *element information items* amongst its [children]. The [namespace name] of such *element information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/soap".

5.8.5 Mapping from XML Representation to Component Properties

See Table 5-5 [p.33] .

Table 5-5. Mapping from XML Representation to SOAP Module component-related Properties

Property	Value
{soap modules [p.31] }	The set of SOAP Module [p.31] components corresponding to all the `module` *element information item* in the [children] of the `binding` , `operation` , `fault` , `input` , `output` , `infault` , `outfault` *element information items*, if any.
{ref [p.31] }	The actual value of the `ref` *attribute information item*.
{required [p.32] }	The actual value of the `required` *attribute information item* if present, otherwise "false".
{parent [p.32] }	The Binding, Binding Operation, Binding Message Reference, Binding Fault or Binding Fault Reference component corresponding to the `binding` , `operation` , `fault` , `input` , `output` , `infault` or `outfault` *element information item* in [parent].

5.8.6 IRI Identification Of A SOAP Module component

WSDL Version 2.0 Part 1: Core Language [*WSDL 2.0 Core Language [p.70]*] defines a fragment identifier syntax for identifying components of a WSDL 2.0 document.

A SOAP Module [p.31] component can be identified using the *wsdl.extension* XPointer Framework scheme:

```
wsdl.extension(http://www.w3.org/2006/01/wsdl/soap,
wsoap.module(parent/ref))
```

1. `parent` is the pointer part of the {parent [p.32] } component, as specified in WSDL Version 2.0 Part 1: Core Language.
2. `ref` is the value of the {ref [p.31] } property of the component.

5.9 Declaring SOAP Header Blocks

5.9.1 Description

SOAP allows the use of header blocks in the header part of the message. This binding extension allows users to declare the SOAP header blocks in use on a per message and on a per fault basis.

5.9.2 Relationship to WSDL Component Model

The SOAP Header Blocks binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {soap headers} OPTIONAL. A set of SOAP Header Block [p.34] components as defined in **5.9.3 SOAP Header Block component** [p.34] , to the Binding Message Reference component.
- Similarly, {soap headers} OPTIONAL, to the Binding Fault component.

5.9.3 SOAP Header Block component

A SOAP Header Block [p.34] component describes an abstract piece of header data (SOAP header block) that is associated with the exchange of messages between the communicating parties. The presence of a SOAP Header Block [p.34] component in a WSDL description indicates that the service supports headers and MAY require a Web service consumer/client that interacts with the service to use the described header block. Zero or one such header block may be used.

The properties of the SOAP Header Block component are as follows:

- {element declaration} REQUIRED. A *xs:QName*, a reference to an XML element declaration in the {element declarations} property of the Description component. This XML element declaration represents a SOAP header block.
- {mustUnderstand} REQUIRED. A *xs:boolean*. When its value is "true", the SOAP header block MUST be decorated with a SOAP `mustUnderstand` *attribute information item* with a value of "true"; if so, it is an error for the XML element declaration referenced by the {element declaration [p.34] } property not to allow this SOAP `mustUnderstand` *attribute information item*. Otherwise, no additional constraint is placed on the presence and value of a SOAP `mustUnderstand` *attribute information item*.

- {required} REQUIRED. A *xs:boolean* indicating if the SOAP header block is required. If the value is "true", then the SOAP header block MUST be included in the message. If it is "false", then the SOAP header block MAY be included.
- {parent} REQUIRED. The Binding Fault or Binding Message Reference component component that contains this component in its {soap headers [p.34] } property.

5.9.4 XML Representation

```
<description>
  <binding name="xs:NCName" type="http://www.w3.org/2006/01/wsdl/soap" >
    <fault ref="xs:QName" >
      <wsoap:header element="xs:QName" mustUnderstand="xs:boolean"?
                  required="xs:boolean"? >
        <documentation />*
      </wsoap:header>*
      ...
    </fault>*
    <operation ref="xs:QName" >
      <input messageLabel="xs:NCName"?>
        <wsoap:header ... />*
        ...
      </input>*
      <output messageLabel="xs:NCName"?>
        <wsoap:header ... />*
        ...
      </output>*
    </operation>*
  </binding>
</description>
```

The XML representation for a SOAP Header Block [p.34] component is an *element information item* with the following Infoset properties:

- A [local name] of `header`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/soap"
- One or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `element` *attribute information item* with the following Infoset properties:
    - A [local name] of `element`
    - A [namespace name] which has no value
    - A type of *xs:QName*
  - An OPTIONAL `mustUnderstand` *attribute information item* with the following Infoset properties:

        - A [local name] of `mustUnderstand`
        - A [namespace name] which has no value
        - A type of *xs:boolean*
    - An OPTIONAL `required` *attribute information item* with the following Infoset properties:
        - A [local name] of `required`
        - A [namespace name] which has no value
        - A type of *xs:boolean*
    - Zero or more namespace qualified *attribute information items*. The [namespace name] of such *attribute information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/soap".
- Zero or more *element information item* amongst its [children], in order, as follows:
    1. Zero or more `documentation` *element information items* as defined in [*WSDL 2.0 Core Language [p.70]*].
    2. Zero or more namespace-qualified *element information items* amongst its [children]. The [namespace name] of such *element information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/soap".

5.9.5 Mapping XML Representation to Component Properties

See Table 5-6 [p.36] .

Table 5-6. Mapping from XML Representation to SOAP Header Block component-related Properties

Property	Value
{soap headers [p.34] }	The set of SOAP Header Block [p.34] components corresponding to all the `header` *element information item* in the [children] of the `fault` , `input` or `output` *element information item*, if any.
{element declaration [p.34] }	The element declaration from the {element declarations} resolved to by the value of the `element` *attribute information item*. It is an error for the `element` *attribute information item* to have a value and that value does not resolve to a global element declaration from the {element declarations} property of the Description component.
{mustUnderstand [p.34] }	The actual value of the `mustUnderstand` *attribute information item* if present, otherwise "false".
{required [p.35] }	The actual value of the `required` *attribute information item* if present, otherwise "false".
{parent [p.35] }	The Binding Fault or Binding Message Reference component corresponding to the `fault` , `input` or `output` *element information item* in [parent].

5.9.6 IRI Identification Of A SOAP Header Block component

WSDL Version 2.0 Part 1: Core Language [*WSDL 2.0 Core Language [p.70]*] defines a fragment identifier syntax for identifying components of a WSDL 2.0 document.

A SOAP Header Block [p.34] component can be identified using the *wsdl.extension* XPointer Framework scheme:

```
wsdl.extension(http://www.w3.org/2006/01/wsdl/soap,
wsoap.header(parent/namespace#name))
```

1. *`parent`* is the pointer part of the {parent [p.35] } component, as specified in WSDL Version 2.0 Part 1: Core Language.
2. *`namespace`* is the {element declaration [p.34] } property value’s namespace URI.
3. *`name`* is the {element declaration [p.34] } property value’s local name.

5.10 WSDL SOAP 1.2 Binding

5.10.1 Identifying a WSDL SOAP 1.2 Binding

A WSDL SOAP Binding is identified as a SOAP 1.2 binding by assigning the value "1.2" to the {soap version [p.26] } property of the Binding component.

5.10.2 Description

The WSDL SOAP 1.2 binding extension defined in this section is an extension of the SOAP binding defined in section **5. WSDL SOAP Binding Extension** [p.22] to enable Web Service applications to use SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework [p.69]*].

The WSDL SOAP 1.2 binding extension supports the SOAP 1.2 HTTP binding defined by the [*SOAP 1.2 Part 2: Adjuncts [p.69]*] specification. This is indicated by assigning the URI "http://www.w3.org/2003/05/soap/bindings/HTTP/" (as defined by [*SOAP 1.2 Part 2: Adjuncts [p.69]*]) to the {soap underlying protocol [p.27] } property. Other values MAY be used for this property in conjunction with the SOAP 1.2 binding extension defined by this specification provided that the semantics of such protocols are consistent with this binding extension.

Default rules in section **5.10.3 SOAP 1.2 Binding Rules** [p.38] define the relationship between SOAP message exchange patterns defined in [*SOAP 1.2 Part 2: Adjuncts [p.69]*] and WSDL message exchange patterns defined in section **2. Predefined Message Exchange Patterns** [p.9] .

When the SOAP Message Exchange Pattern is the SOAP 1.2 Response MEP and the underlying protocol is HTTP, the Binding Operation may use the {http location [p.47] } property defined in section **6.5 Binding Operations** [p.47] . When this property is present on the Binding Operation component, the Endpoint component also follows the rules for constructing the address from the {address} property and the {http location [p.47] } property values.

5.10.3 SOAP 1.2 Binding Rules

These binding rules are applicable to SOAP 1.2 bindings.

- *SOAP Action Feature.* The value of the SOAP Action Feature for the initial message of the message exchange pattern of the Interface Operation bound is specified by the {soap action [p.29] } property of this Binding Operation component. If the Binding Operation component does NOT have a {soap action [p.29] } property defined, then the SOAP Action Feature (see [*SOAP 1.2 Part 2: Adjuncts [p.69]*]) has NO value. Otherwise, its value is the value of the SOAP Action Feature for the initial message of the message exchange pattern.

- *SOAP MEP Selection.* For a given Interface Operation component, if there is a Binding Operation component whose {interface operation} property matches the component in question and its {soap mep [p.29] } property has a value, then SOAP MEP is the value of the {soap mep [p.29] } property. Otherwise, the SOAP MEP is the value of the Binding component's {soap mep default [p.29] }, if any. Otherwise, if the Interface Operation component's {message exchange pattern} property has the value "http://www.w3.org/2006/01/wsdl/in-out", then the SOAP MEP is the URI "http://www.w3.org/2003/05/soap/mep/request-response/" identifying the SOAP Request-Response Message Exchange Pattern as defined in [*SOAP 1.2 Part 2: Adjuncts [p.69]*]. Otherwise (i.e. if the Interface Operation component has any other value for the {message exchange pattern} property), it is an ERROR.

Editorial note: One-way MEP defaulting	
The Web Services Description Working Group would like to add a rule here defaulting to a standardized SOAP 1.2 one-way MEP for one-way operations if one becomes available. Feedback is sought on this topic.	

- *HTTP Method Selection.* This default binding rule is applicable when the value of the {soap underlying protocol [p.27] } property of the Binding component is "http://www.w3.org/2003/05/soap/bindings/HTTP/". If the SOAP MEP selected as specified above has the value "http://www.w3.org/2003/05/soap/mep/request-response/" then the HTTP method used is "POST". If the SOAP MEP selected has the value "http://www.w3.org/2003/05/soap/mep/soap-response/" then the HTTP method used is "GET".

- *HTTP IRI Generation.* This default binding rule is applicable when the value of the {soap underlying protocol [p.27] } property of the Binding component is "http://www.w3.org/2003/05/soap/bindings/HTTP/". If the SOAP MEP selected is "http://www.w3.org/2003/05/soap/mep/soap-response/" then the value of the SOAP "http://www.w3.org/2003/05/soap/mep/ImmediateDestination" property MUST be generated using the HTTP binding extension's rules for generating a IRI for HTTP GET (see **6.8.2 Serialization as application/x-www-form-urlencoded** [p.58]). The input serialization format of `application/x-www-form-urlencoded` is the only supported serialization format for HTTP GET in the SOAP Response Message Exchange Pattern.

5.10.4 Binding WSDL 2.0 MEPs to SOAP 1.2 MEPs

This section describes the relationship between WSDL components and SOAP 1.2 MEP properties as described in [*SOAP 1.2 Part 2: Adjuncts [p.69]*].

5.10.4.1 Using SOAP Request-Response

When using the WSDL "http://www.w3.org/2006/01/wsdl/in-out" message exchange pattern bound to a SOAP "http://www.w3.org/2003/05/soap/mep/request-response/" MEP (as would be the case for a usual SOAP-over-HTTP In-Out operation), this section describes the relationships. Extensions (such as [*WSA 1.0 Core [p.70]*]) MAY alter these mappings.

5.10.4.1.1 The Client

As the client, the property "http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role" takes the value "RequestingSOAPNode".

The SOAP "http://www.w3.org/2003/05/soap/mep/ImmediateDestination" property takes the value of the WSDL {address} property of the Endpoint component.

The WSDL "In" message is mapped to the SOAP "http://www.w3.org/2003/05/soap/mep/OutboundMessage" property.

The WSDL "Out" message maps to the SOAP "http://www.w3.org/2003/05/soap/mep/InboundMessage" property.

5.10.4.1.2 The Service

As the service, the property "http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role" takes the value "RespondingSOAPNode".

The WSDL "In" message is mapped to the SOAP "http://www.w3.org/2003/05/soap/mep/InboundMessage" property.

The WSDL "Out" message maps to the SOAP "http://www.w3.org/2003/05/soap/mep/OutboundMessage" property.

5.10.4.2 Using SOAP-Response

When using the WSDL "http://www.w3.org/2006/01/wsdl/in-out" message exchange pattern bound to a "http://www.w3.org/2003/05/soap/mep/soap-response/" SOAP MEP, this section describes the relationships.

5.10.4.2.1 The Client

As the client, the property "http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role" takes the value "RequestingSOAPNode".

The SOAP "http://www.w3.org/2003/05/soap/mep/ImmediateDestination" property takes the value of the WSDL {address} property, modified by the {http location [p.47] } property following the rules described in section **6.8.2 Serialization as application/x-www-form-urlencoded** [p.58] .

The SOAP "http://www.w3.org/2003/05/soap/mep/OutboundMessage" property has no value.

The WSDL "Out" message maps to the SOAP "http://www.w3.org/2003/05/soap/mep/InboundMessage" property.

5.10.4.2.2 The Service

As the service, the property "http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role" takes the value "RespondingSOAPNode".

The WSDL "In" message is constructed from the destination URI as per the rules in section **6.8.2 Serialization as application/x-www-form-urlencoded** [p.58] .

The WSDL "Out" message maps to the SOAP "http://www.w3.org/2003/05/soap/mep/OutboundMessage" property.

5.11 Conformance

An *element information item* whose namespace name is "http://www.w3.org/2006/01/wsdl" and whose local part is `description` conforms to this binding extension specification if the *element information item*s and *attribute information item*s whose namespace is http://www.w3.org/2006/01/wsdl/soap conform to the XML Schema for that element or attribute as defined by this specification and additionally adheres to all the constraints contained in this specification.

6. WSDL HTTP Binding Extension

The HTTP binding extension described in this section is an extension for [*WSDL 2.0 Core Language [p.70]*] to enable Web Services applications to use HTTP 1.1 [*IETF RFC 2616 [p.68]*] (as well as other versions of HTTP) and HTTPS [*IETF RFC 2818 [p.69]*]. This binding extension extends WSDL 2.0 by adding properties to the component model defined in [*WSDL 2.0 Core Language [p.70]*]. In addition an XML Infoset representation for these additional properties is provided, along with a mapping from that representation to the various component properties.

As allowed in [*WSDL 2.0 Core Language [p.70]*], a Binding component MAY exist without indicating a specific Interface component that it applies to. In this case there MUST NOT be any Binding Operation or Binding Fault components present in the Binding component.

The HTTP binding extension is designed with the objective of minimizing what needs to be explicitly declared for common cases. This is achieved by defining a set of default rules which apply for all Interface Operation components of an Interface component, unless specifically overridden on a per Interface Operation basis. Thus, if a given Interface Operation component is not referred to specifically, then all the default rules apply for that component. That is, per the requirements of [*WSDL 2.0 Core Language [p.70]*] all operations of an Interface component are bound by an HTTP binding.

[Definition: The internal tree representation of an input, output or fault message is called an **instance data**, and is constrained by the schema definition associated the message: the XML element referenced in the {element declaration} property of the Interface Message Reference component for input and output messages (unless the {message content model} is "#any"), and in the {element declaration} property of an Interface Fault component for faults.]

6.1 Identifying the use of the HTTP Binding

A Binding component (defined in [*WSDL 2.0 Core Language [p.70]*]) is identified as an HTTP binding by assigning the value "http://www.w3.org/2006/01/wsdl/http" to the {type} property of the Binding component.

6.2 HTTP Syntax Summary (Non-Normative)

```
<description>
  <binding name="xs:NCName" interface="xs:QName"?
           type="http://www.w3.org/2006/01/wsdl/http"
           whttp:methodDefault="xs:string"?
           whttp:queryParameterSeparatorDefault="xs:string"?
```

```
        whttp:cookies="xs:boolean"?
        whttp:version="xs:string"?
        whttp:transferCodingDefault="xs:string"? >
<documentation />?

 <fault ref="xs:QName"
        whttp:code="union of xs:int, xs:token"?
        whttp:transferCoding="xs:string"? >
   <documentation />*
   <whttp:header name="xs:string" type="xs:QName"
                 required="xs:boolean"? >
     <documentation />*
   </whttp:header>*
   [ <feature /> | <property /> ]*
 </fault>*

 <operation ref="xs:QName"
            whttp:location="xs:anyURI"?
            whttp:method="xs:string"?
            whttp:inputSerialization="xs:string"?
            whttp:outputSerialization="xs:string"?
            whttp:faultSerialization="xs:string"?
            whttp:transferCodingDefault="xs:string"? >
       <documentation />*

   <input messageLabel="xs:NCName"?
          whttp:transferCoding="xs:string? >
     <documentation />*
     <whttp:header ... />*
     [ <feature /> | <property /> ]*
   </input>*

   <output messageLabel="xs:NCName"?
           whttp:transferCoding="xs:string? >
     <documentation />*
     <whttp:header ... />*
     [ <feature /> | <property /> ]*
   </output>*

   <infault ref="xs:QName"
              messageLabel="xs:NCName"? >
     <documentation />*
     [ <feature /> | <property /> ]*
   </infault>*

   <outfault ref="xs:QName"
             messageLabel="xs:NCName"? >
     <documentation />*
     [ <feature /> | <property /> ]*
   </outfault>*

   [ <feature /> | <property /> ]*

 </operation>*

 [ <feature /> | <property /> ]*
```

```
  </binding>

  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"?
              whttp:authenticationType="xs:token"?
              whttp:authenticationRealm="xs:string"? >
      <documentation />*
      [ <feature /> | <property /> ]*
    </endpoint>
    [ <feature /> | <property /> ]*
  </service>
</description>
```

6.3 HTTP Binding Rules

6.3.1 HTTP Method Selection

When formulating the HTTP message to be transmitted, the HTTP request method used MUST be the following:

- For a given Interface Operation component, if there is a Binding Operation component whose {interface operation} property matches the component in question and its {http method [p.48] } property has a value, then the value of the {http method [p.48] } property.
- Otherwise, the value of the Binding component's {http method default [p.47] }, if any.
- Otherwise, if a {safety [p.15] } property as defined in **3.1 Operation safety** [p.15] is present on the bound Interface Operation component and has a value of "true", the value "GET".
- Otherwise, it is an ERROR.

6.3.2 Payload Construction And Serialization Format

When formulating the HTTP message to be transmitted, the contents of the payload (i.e. the contents of the HTTP message body) MUST be what is defined by the corresponding Interface Message Reference or Interface Fault components, serialized as specified by the serialization format [p.43] used.

[Definition: The **serialization format** is a media type token ("type/subtype"). It identifies rules to serialize a message in an HTTP message. Its value follows the following rules. The HTTP request serialization format MUST be in the media type range specified by the {http input serialization [p.48] } property. The HTTP response serialization format MUST be in the media type range specified by the {http output serialization [p.48] } property. The HTTP serialization format of a fault MUST be in the media type range specified by the {http fault serialization [p.48] } property. The concept of media type range is defined in Section 14.1 of [*IETF RFC 2616 [p.68]*]. The serialization format MAY have **associated media type parameters** (specified with the `parameter` production of `media-range` in Section 14.1 of [*IETF RFC 2616 [p.68]*].]

Section **6.8 Serialization Format of Instance Data** [p.55] defines serialization formats supported by this binding extension along with their constraints.

- Interface Message Reference component:
  - If the value of the {message content model} property of the Interface Message Reference bound is "#any" or "#element", the serialization of the instance data is specified as defined in section **6.3.2.1 Serialization rules for XML messages** [p.44] .
  - If the value is "#none" then the payload MUST be empty and the value of the corresponding serialization property ({http input serialization [p.48] } or {http output serialization [p.48] }) is ignored.
  - If the value is "#other" then the serialization format and its associated media type parameters, if any [p.43] specifies the value of the HTTP `Content-Type` entity-header field as defined in section 14.17 of [*IETF RFC 2616 [p.68]*]. The serialization of the payload is undefined.
- Interface Fault component: the serialization of the instance data is specified as defined in section **6.3.2.1 Serialization rules for XML messages** [p.44] .

If the Interface Message Reference component or the Interface Fault component is declared using a non-XML type system (as considered in the Types section of [*WSDL 2.0 Core Language [p.70]*]) then additional binding rules MUST be defined to indicate how to map those components into the HTTP envelope.

6.3.2.1 Serialization rules for XML messages

The serialization rules for messages whose {message content model} is either "#element" or "#any" and for fault messages are as follows:

- If the serialization format [p.43] is "application/x-www-form-urlencoded", then the serialization of the instance data [p.41] is defined by section **6.8.2 Serialization as application/x-www-form-urlencoded** [p.58] .
- If the serialization format [p.43] is "multipart/form-data", then the serialization of the instance data [p.41] is defined by section **6.8.4 Serialization as multipart/form-data** [p.62] .
- If the serialization format [p.43] is "application/xml", then the serialization of the instance data [p.41] is defined by section **6.8.3 Serialization as application/xml** [p.61] .
- Otherwise, then the serialization of the instance data [p.41] is defined by section **6.8.3 Serialization as application/xml** [p.61] with the following additional rule: the value of the HTTP `Content-Type` entity-header field is the value of the serialization format and its associated media type parameters, if any [p.43] .

6.3.3 Default input and output serialization format

Section Table 6-1 [p.45] defines the default values for the GET, POST, PUT and DELETE values of the HTTP method as selected in section **6.3.1 HTTP Method Selection** [p.43] .

Table 6-1. Default values for GET, POST, PUT and DELETE

HTTP Method	Default Input Serialization	Default Output Serialization
Selected in 6.3.1 HTTP Method Selection [p.43]	**{http input serialization [p.48] }**	**{http output serialization [p.48] }**
GET	`application/x-www-form-urlencoded`	`application/xml`
POST	`application/xml`	`application/xml`
PUT	`application/xml`	`application/xml`
DELETE	`application/x-www-form-urlencoded`	`application/xml`

Note:

The `application/x-www-form-urlencoded` serialization format places constraints on the XML Schema definition of the {element declaration} property of the Interface Message Reference components of the Interface Operation component bound (see **6.8.2 Serialization as application/x-www-form-urlencoded** [p.58]).

The default value for the {http input serialization [p.48] } and {http output serialization [p.48] } properties for any other HTTP method selected is `application/xml`.

Mechanisms other than setting the serialization properties MAY modify the serialization format of the instance data [p.41] corresponding to the message. An example of such modification is the WSDL SOAP Binding HTTP IRI Serialization rules specified in **5.3 SOAP Binding Rules** [p.25] . This binding extension specifies that the SOAP-Response Message Exchange Pattern ([*SOAP 1.2 Part 2: Adjuncts [p.69]*], Section 6.3) only supports input message serialization as `application/x-www-form-urlencoded`. Other examples of such mechanisms are other message exchange patterns or binding extensions.

6.3.4 HTTP Header Construction

If the {http headers [p.51] } property as defined in section **6.6 Declaring HTTP Headers** [p.51] exists and is not empty in a Binding Message Reference or Binding Fault component, HTTP headers conforming to each HTTP Header [p.52] component contained in this {http headers [p.51] } property MAY be serialized as follows:

- The HTTP header field name used is the value of the {name [p.52] } property of the HTTP Header [p.52] component. If an HTTP header field corresponding to the value of the {name [p.52] } property is set by a mechanism other than the HTTP binding, such as the HTTP stack or another feature, then an error MUST be raised.

- The HTTP header field value, whose XML Schema type is declared by the {type definition [p.52] } property of the HTTP Header [p.52] component, is serialized following the rules of the `field-value` production of section 4.2 of [*IETF RFC 2616 [p.68]*].

If the value of an HTTP Header [p.52] component's {required [p.52] } property is "true", the inclusion of this HTTP header field is REQUIRED, otherwise it is OPTIONAL.

6.4 Specifying the HTTP Version

6.4.1 Description

Every Binding component MUST indicate what version of HTTP is in use for the operations of the interface that this binding applies to.

By default, HTTP/1.1 [*IETF RFC 2616 [p.68]*] is used.

6.4.2 Relationship to WSDL Component Model

The HTTP binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http version} REQUIRED. A *xs:string* to the Binding component. The value of this property follows the "<major>.<minor>" numbering scheme defined in section 3.1 of Hypertext Transfer Protocol -- HTTP/1.1 [*IETF RFC 2616 [p.68]*].

6.4.3 XML Representation

```
<description>
  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI"
           whttp:version="xs:string"? >
  </binding>
</description>
```

The XML representation for specifying the HTTP version is an optional *attribute information item* with the following Infoset properties:

- A [local name] of `version`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
- A type of *xs:string* whose pattern facet is *"[0-9]+\.[0-9]+"*.

6.4.4 Mapping from XML Representation to Component Properties

See Table 6-2 [p.47] .

Table 6-2. Mapping from XML Representation to Binding component Extension Properties

Property	Value
{http version [p.46] }	The actual value of the `whttp:version` *attribute information item*, if present, otherwise "1.1".

6.5 Binding Operations

6.5.1 Description

This binding extension specification provides a binding to HTTP of Interface Operation components whose {message exchange pattern} property has the value "http://www.w3.org/2006/01/wsdl/in-only", "http://www.w3.org/2006/01/wsdl/robust-in-only" or "http://www.w3.org/2006/01/wsdl/in-out". This HTTP binding extension MAY be used with other message exchange patterns such as outbound message exchange patterns, provided that additional semantics are defined, such as with an extension or with a Feature.

Each of the supported message exchange patterns involves one to two messages or faults being exchanged. The first is transmitted using an HTTP request, and the second is transmitted using the corresponding HTTP response. In cases where only one message is being sent, the message body of the HTTP response MUST be empty.

For every Binding Operation component corresponding to such Interface Operation components, this binding extension specification allows the user to indicate the HTTP method to use, the input, output and fault serialization, and the location of the bound operation.

6.5.2 Relationship to WSDL Component Model

The HTTP binding extension adds the following properties to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http location} OPTIONAL. A *xs:anyURI*, to the Binding Operation component. This IRI is combined with the base IRI specified in the {address} property of the Endpoint component to form the full IRI for the HTTP request to invoke the operation. It MUST contain an absolute or a relative IRI, i.e. it MUST NOT include a fragment identifier in the IRI. Input serializations may define additional processing rules to be applied to the value of {http location [p.47] } before combining it with the {address} property of the endpoint element to form the HTTP request IRI. For example, the three serialization formats defined in section **6.8 Serialization Format of Instance Data** [p.55] define a syntax to use the {http location [p.47] } as a template using elements of the instance data.

  If the resulting IRI uses the `https` scheme, then HTTP over TLS [*IETF RFC 2818 [p.69]*] is used to send the HTTP request.

- {http method default} OPTIONAL. A *xs:string*, to the Binding component, indicating the default value for the HTTP Request Method for all the Interface Operation components of any Interface component that uses this Binding component.

- {http method} OPTIONAL. A *xs:string*, to the Binding Operation component, indicating the value for the HTTP Request Method for this specific Binding Operation.
- {http input serialization} REQUIRED. A *xs:string*, to the Binding Operation component, indicating allowed serialization rules of the HTTP Request message for this specific operation, as described in section **6.5.3 Specification of serialization rules allowed** [p.48] .
- {http output serialization} REQUIRED. A *xs:string*, to the Binding Operation component, indicating allowed serialization rules of the HTTP Response message for this specific operation, as described in section **6.5.3 Specification of serialization rules allowed** [p.48] .
- {http fault serialization} REQUIRED. A *xs:string*, to the Binding Operation component, indicating allowed serialization rules of the HTTP Response message for this specific operation in case a fault is returned, as described in section **6.5.3 Specification of serialization rules allowed** [p.48] .
- {http query parameter separator default} REQUIRED. A *xs:string*, to the Binding component, indicating the default query parameter separator character for all the Interface Operation components of any Interface component that uses this Binding component.
- {http query parameter separator} OPTIONAL. A *xs:string*, to the Binding Operation component, indicating the query parameter separator character for this Binding Operation component.

6.5.3 Specification of serialization rules allowed

The value of the {http input serialization [p.48] }, {http output serialization [p.48] } and {http fault serialization [p.48] } properties is similar to the value allowed for the `Accept` HTTP header defined by the HTTP 1.1 specification, Section 14.1 (see [*IETF RFC 2616 [p.68]*]) and MUST follow the production rules defined in that section except for the following:

1. The prefix `"Accept:"` MUST NOT be used.
2. The rule `qdtext` is changed from:

   ```
   qdtext = <any TEXT except<">>
   ```

   to:

   ```
   qdtext = <any CHAR except<">>
   ```

   This change is made to disallow non-US-ASCII OCTETs.

These properties allow to indicate the range of media types and/or associated parameters with which an instance MAY be serialized. The value of the serialization format [p.43] used for a message is a media type which MUST be covered by this range. Users of this *attribute information item* are urged to avoid using wild cards (for example, "application/*") as it may lead to interoperability problems.

The use of {http input serialization [p.48] }, {http output serialization [p.48] } and {http fault serialization [p.48] } is specified in section **6.3.2 Payload Construction And Serialization Format** [p.43] .

6.5.4 XML Representation

```
<description>
 <binding whttp:methodDefault="xs:string"?
          whttp:queryParameterSeparatorDefault="xs:string"? >
   <operation ref="xs:QName"
              whttp:location="xs:anyURI"?
              whttp:method="xs:string"?
              whttp:inputSerialization="xs:string"?
              whttp:outputSerialization="xs:string"?
              whttp:faultSerialization="xs:string"?
              whttp:queryParameterSeparator="xs:string"? >
  </operation>
 </binding>
</description>
```

The XML representation for binding an Operation are six *attribute information item*s with the following Infoset properties:

- An OPTIONAL `location` *attribute information item* with the following Infoset properties:
  - A [local name] of `location`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:anyURI*
- An OPTIONAL `method` *attribute information item* with the following Infoset properties:
  - A [local name] of `method`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*
- An OPTIONAL `inputSerialization` *attribute information item* with the following Infoset properties:
  - A [local name] of `inputSerialization`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*
- An OPTIONAL `outputSerialization` *attribute information item* with the following Infoset properties:
  - A [local name] of `outputSerialization`

  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*
- An OPTIONAL `faultSerialization` *attribute information item* with the following Infoset properties:
  - A [local name] of `faultSerialization`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*
- An OPTIONAL `queryParameterSeparator` *attribute information item* with the following Infoset properties:
  - A [local name] of `queryParameterSeparator`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string* whose length facet value is *"1"*

The following *attribute information item*s for the `binding` *element information item* are defined:

- An OPTIONAL `methodDefault` *attribute information item* with the following Infoset properties:
  - A [local name] of `methodDefault`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*
- An OPTIONAL `queryParameterSeparatorDefault` *attribute information item* with the following Infoset properties:
  - A [local name] of `queryParameterSeparatorDefault`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string* whose length facet value is *"1"*

6.5.5 Mapping from XML Representation to Component Properties

See Table 6-3 [p.50] .

Table 6-3. Mapping from XML Representation to Binding Operation component Extension Properties

Property	Value
{http location [p.47] }	The actual value of the `whttp:location` *attribute information item*, if present.
{http method default [p.47] }	The actual value of the `whttp:methodDefault` *attribute information item*, if present.
{http method [p.48] }	The actual value of the `whttp:method` *attribute information item*, if present.
{http input serialization [p.48] }	The actual value of the `whttp:inputSerialization` *attribute information item*, if present; otherwise, the default value as defined in **6.3 HTTP Binding Rules** [p.43] .
{http output serialization [p.48] }	The actual value of the `whttp:outputSerialization` *attribute information item*, if present; otherwise, the default value as defined in **6.3 HTTP Binding Rules** [p.43] .
{http fault serialization [p.48] }	The actual value of the `whttp:faultSerialization` *attribute information item*, if present; otherwise "application/xml".
{http query parameter separator default [p.48] }	The actual value of the `whttp:queryParameterSeparatorDefault` *attribute information item*, if present; otherwise, "&".
{http query parameter separator [p.48] }	The actual value of the `whttp:queryParameterSeparator` *attribute information item*, if present.

6.6 Declaring HTTP Headers

6.6.1 Description

HTTP allows the use of headers in messages. This binding extension allows users to declare the HTTP headers in use on a per message and on a per fault basis.

6.6.2 Relationship to WSDL Component Model

The HTTP Header binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http headers} OPTIONAL. A set of HTTP Header [p.52] components as defined in **6.6.3 HTTP Header component** [p.52] , to the Binding Message Reference component.

- Similarly, {http headers} OPTIONAL, to the Binding Fault component.

It is an ERROR for a Binding Message Reference or a Binding Fault component's {http headers [p.51] } property to contain multiple HTTP Header [p.52] components with the same {name [p.52] } property.

6.6.3 HTTP Header component

A HTTP Header [p.52] component describes an abstract piece of header data (HTTP header field) that is associated with the exchange of messages between the communicating parties. The presence of a HTTP Header [p.52] component in a WSDL description indicates that the service support headers and MAY require a Web service consumer/client that interacts with the service to use the described header field. Zero or one such header field may be used.

The properties of the HTTP Header component are as follows:

- {name} REQUIRED. A *xs:string* whose pattern facet is "[!#-'*+\-.0-9A-Z^-z|~]+", the name of the HTTP header field. The value of this property follows the `field-name` production rules as specified in section 4.2 of [*IETF RFC 2616 [p.68]*].

- {type definition} REQUIRED. A *xs:QName*, being a reference to a Type Definition component in the {type definitions} property of the Description component constraining the value of the HTTP header field. This type MUST be a simple type.

- {required} REQUIRED. A *xs:boolean* indicating if the HTTP header field is required. If the value is "true", then the HTTP header field MUST be included in the message. If it is "false", then the HTTP header field MAY be included.

- {parent} REQUIRED. The Binding Fault or Binding Message Reference component component that contains this component in its {http headers [p.51] } property.

6.6.4 XML Representation

```
<description>
  <binding name="xs:NCName" type="http://www.w3.org/2006/01/wsdl/http" >
    <fault ref="xs:QName">
      <whttp:header name="xs:QName" type="xs:QName"
                    required="xs:boolean"? >
        <documentation />*
      </whttp:header>*
      ...
    </fault>*
    <operation ref="xs:QName" >
      <input messageLabel="xs:NCName"?>
        <whttp:header ... />*
        ...
      </input>*
      <output messageLabel="xs:NCName"?>
        <whttp:header ... />*
        ...
```

```
      </output>*
    </operation>*
  </binding>
</description>
```

The XML representation for a HTTP Header [p.52] component is an *element information item* with the following Infoset properties:

- A [local name] of `header`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
- One or more *attribute information items* amongst its [attributes] as follows:
    - A REQUIRED `name` *attribute information item* with the following Infoset properties:
        - A [local name] of `name`
        - A [namespace name] which has no value
        - A type of *xs:string* whose pattern facet is _"[!#-'*+\-.0-9A-Z^-z|~]+"_.
    - A REQUIRED `type` *attribute information item* with the following Infoset properties:
        - A [local name] of `type`
        - A [namespace name] which has no value
        - A type of *xs:QName*
    - An OPTIONAL `required` *attribute information item* with the following Infoset properties:
        - A [local name] of `required`
        - A [namespace name] which has no value
        - A type of *xs:boolean*
    - Zero or more namespace qualified *attribute information items*. The [namespace name] of such *attribute information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/http".
- Zero or more *element information item* amongst its [children], in order, as follows:
    1. Zero or more `documentation` *element information items* as defined in [*WSDL 2.0 Core Language [p.70]*].
    2. Zero or more namespace-qualified *element information items* amongst its [children]. The [namespace name] of such *element information items* MUST NOT be "http://www.w3.org/2006/01/wsdl" and MUST NOT be "http://www.w3.org/2006/01/wsdl/http".

6.6.5 Mapping from XML Representation to Component Properties

See Table 6-4 [p.54] .

Table 6-4. Mapping from XML Representation to HTTP Header component-related Properties

Property	Value
{http headers [p.51] }	The set of HTTP Header [p.52] components corresponding to all the `header` *element information item* in the [children] of the `fault` , `input` or `output` *element information item*, if any.
{name [p.52] }	The value of the `name` *attribute information item*.
{type definition [p.52] }	The Type Definition component from the {type definitions} property of the Description component resolved to by the value of the `type` *attribute information item*.
{required [p.52] }	The actual value of the `required` *attribute information item* if present, otherwise "false".
{parent [p.52] }	The Binding Fault or Binding Message Reference component corresponding to the `fault` , `input` or `output` *element information item* in [parent].

6.6.6 IRI Identification Of A HTTP Header component

WSDL Version 2.0 Part 1: Core Language [*WSDL 2.0 Core Language [p.70]*] defines a fragment identifier syntax for identifying components of a WSDL 2.0 document.

An HTTP Header [p.52] component can be identified using the *wsdl.extension* XPointer Framework scheme:

```
wsdl.extension(http://www.w3.org/2006/01/wsdl/http,
whttp.header(parent/name))
```

1. *parent* is the pointer part of the {parent [p.52] } component, as specified in WSDL Version 2.0 Part 1: Core Language.

2. *name* is the {name [p.52] } property value.

6.7 Specifying HTTP Error Code for Faults

6.7.1 Description

For every Interface Fault component contained in an Interface component, an HTTP error code MAY be defined. It represents the error code that will be used by the service in case the fault needs to be returned.

The fault definition SHOULD NOT go against the definition of the HTTP error codes, as specified in section 8 of [*IETF RFC 3205 [p.69]*].

6.7.2 Relationship to WSDL Component Model

The HTTP Fault binding extension adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http error status code} REQUIRED. A union of *xs:int* and *xs:token* where the allowed token value is "#any", to the Binding Fault component. An integer value of this property identifies the error Status-Code as defined by [*IETF RFC 2616 [p.68]*] that the service will use in case the fault is returned. If the value of this property is "#any", no claim is made by the service.

6.7.3 XML Representation

```
<description>
  <binding >
    <fault ref="xs:QName"
           whttp:code="union of xs:int, xs:token"? >
    </fault>*
  </binding>
</description>
```

The XML representation for binding an HTTP Fault are two *attribute information item*s with the following Infoset properties:

- a `code` OPTIONAL *attribute information item*
    - A [local name] of `code`
    - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
    - A type of union of *xs:int* and *xs:token* where the allowed token value is "#any"

6.7.4 Mapping from XML Representation to Component Properties

See Table 6-5 [p.55] .

Table 6-5. Mapping from XML Representation to Binding Fault component Extension Properties

Property	Value
{http error status code [p.55] }	The actual value of the `whttp:code` *attribute information item*, if present; otherwise "#any".

6.8 Serialization Format of Instance Data

This section specifies three serialization formats defining rules to encode an instance data [p.41] corresponding to an input and output message as an HTTP message. Table 6-6 [p.56] and Table 6-7 [p.56] give an overview of those serialization formats and their constraints. All of them allow serialization of parts of the instance data [p.41] in the HTTP Request IRI, as defined in section **6.8.1 Serialization of the instance data in parts of the HTTP request IRI** [p.57] .

Other serialization formats may be defined. Those MAY place restrictions on the style of the Interface Operation bound.

Table 6-6. Applicability of the serialization formats defined in this section for this HTTP binding

<table>
<tr><th colspan="2" rowspan="3">-</th><th colspan="4">Serialization of the instance data in parts of an HTTP message</th></tr>
<tr><th rowspan="2">In the request URI</th><th colspan="3">In the message body</th></tr>
<tr><th>application/x-www-form-urlencoded</th><th>multipart/form-data</th><th>application/xml</th></tr>
<tr><td rowspan="2">HTTP request (input message)</td><td>Without message body: GET, DELETE, …</td><td>All, some or none</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>With message body: POST, PUT, …</td><td>All, some or none</td><td>Remainder</td><td>All</td><td>All</td></tr>
<tr><td colspan="2">HTTP response (output message)</td><td>-</td><td>-</td><td>-</td><td>All</td></tr>
</table>

Table 6-7. Operation styles required for using serialization formats defined below as input serialization

<table>
<tr><th rowspan="3">HTTP Method</th><th colspan="4">Request</th></tr>
<tr><th rowspan="2">Request URI: query parameters or path components</th><th colspan="3">Input serialization</th></tr>
<tr><th>application/x-www-form-urlencoded</th><th>multipart/form-data</th><th>application/xml</th></tr>
<tr><td>Without message body: GET, DELETE, …</td><td>IRI style</td><td>IRI style</td><td>-</td><td>-</td></tr>
</table>

With message body: POST, PUT, …	IRI style, if any data is serialized as path components or query parameters	IRI style	Multipart style	None required

6.8.1 Serialization of the instance data in parts of the HTTP request IRI

Editorial note: URIPath Feedback Requested
The inclusion of elements of the instance data in the path of the request URI, whilst supported by WSDL 1.1, is not supported by XForms 1.0. Hence this mechanism MAY be removed in a future version of this specification. Feedback on this issue from users and implementers is highly encouraged.

This section defines templating rules for the {http location [p.47] } property of the Binding Operation component. It is used by the serialization formats defined in section **6.8 Serialization Format of Instance Data** [p.55] , and MAY be reused by other serialization formats.

With this HTTP binding, part of the instance data for HTTP requests MAY be serialized in the HTTP request IRI, and another part MAY be serialized in the HTTP message body.

If the {style} property of the Interface Operation bound has a value of "http://www.w3.org/2006/01/wsdl/style/iri" as defined in **4.2 IRI Style** [p.20] , and if the {http location [p.47] } property of the Binding Operation component is present, the value of the {http location [p.47] } property component is used as a template which is combined with the {address} property of the endpoint element to form the full IRI to be used in an HTTP request, as specified in section **6.5.2 Relationship to WSDL Component Model** [p.47] .

The resulting IRI MUST be mapped to an URI for use in the HTTP Request as per section 3.1 "Mapping of IRIs to URIs" of the IRI specification [*IETF RFC 3987 [p.69]*]. Additional rules for the serialization of the HTTP request IRI MAY be defined by a serialization format.

6.8.1.1 Construction of the request IRI using the {http location} property

The {http location [p.47] } property, if present, MAY cite local names of elements from the instance data [p.41] of the message to be serialized in request IRI by enclosing the element name within curly braces (e.g. "temperature/{town}"):

- When constructing the request IRI, each pair of curly braces (and enclosed element name) is replaced by the possibly empty single value of the corresponding element. It is an error for this element to carry an `xs:nil` attribute whose value is "true".

- A double curly brace (i.e. "{{" or "}}") MAY be used to include a single, literal curly brace in the request IRI.

An element MUST NOT be cited more than once within the {http location [p.47] } property.

Strings enclosed within single curly braces MUST be element names from the instance data [p.41] of the input message, possibly followed by a slash; any other strings enclosed within single curly braces are a fatal error.

6.8.2 Serialization as "application/x-www-form-urlencoded"

This serialization format is designed to allow a client or Web service to produce an IRI based on the instance data [p.41] of a message and serialize a query string in the HTTP message body as `application/x-www-form-urlencoded`.

It may only be used when binding Interface Operation whose {style} property has a value of "http://www.w3.org/2006/01/wsdl/style/iri" as defined in **4.2 IRI Style** [p.20] , i.e. this serialization format may only be used to serialize the HTTP request corresponding to the initial message of an interface operation.

Because the IRI Style constrains the instance data not to contain multiple children elements declared with the same local name, elements can be serialized in the request IRI with their local names unambiguously.

For the HTTP binding defined in this section (**6. WSDL HTTP Binding Extension** [p.41]), "application/x-www-form-urlencoded" MAY be used as a serialization format [p.43] for an input message (HTTP Request), but MUST NOT be used as a serialization format [p.43] for an output or fault message (HTTP Response).

6.8.2.1 Case of elements cited in the {http location} property

In this serialization, the rules for constructing the HTTP request IRI using elements cited in the {http location [p.47] } property defined in **6.8.1 Serialization of the instance data in parts of the HTTP request IRI** [p.57] apply. Additional rules for constructing the HTTP request IRI follow.

6.8.2.2 Serialization of content of the instance data not cited in the {http location} property

If not all elements from the instance data [p.41] are cited in the {http location [p.47] } property, or if the property is not present on the Binding Operation component, then additional serialization rules apply.

The remainder of the instance data is formatted as a query string as defined in **6.8.2.2.1 Construction of the query string** [p.59] .

If the HTTP method used for the request does not allow a message body, then this query string is serialized as parameters in the request IRI (see **6.8.2.2.3 Serialization in the request IRI** [p.60]), otherwise it is serialized in the message body (see **6.8.2.2.4 Serialization in the message body** [p.61]).

6.8.2.2.1 Construction of the query string

For elements of the instance data not cited in the {http location [p.47] } property, a query string is constructed as follows.

Non-nil elements with a possibly empty single value of the instance data [p.41] not cited are serialized as query parameters in the order they appear in the instance data.

It is an error for the instance data [p.41] to contain elements with an `xs:nil` attribute whose value is "true".

Each parameter pair is separated by the value of the {http query parameter separator [p.48] } property, if present, or the value of the {http query parameter separator default [p.48] } property.

- Uncited elements with single values (non-list) are serialized as a single name-value parameter pair. The name of the parameter is the local name of the uncited element, and the value of the parameter is the value of the uncited element.
- Uncited elements with list values are serialized as one name-value parameter pair per list value. The name of each parameter is the local name of the uncited element, and the value of each parameter is the corresponding value in the list. The order of the list values is preserved.

Example 6-1. Query string generation

The following instance data of an input message

```
<data>
  <town>Fréjus</town>
  <date>2006-01-06</date>
  <unit>C</unit>
</data>
```

with the following value of the {http location [p.47] } property:

```
'temperature/{town}'
```

and the following value of the {http query parameter separator default [p.48] } property:

```
'&'
```

will produce the following query string:

```
date=2006-01-06&unit=C
```

6.8.2.2.2 Controlling the serialization of the query string in the request IRI

This serialization format adds the following property to the WSDL component model:

- {http location ignore uncited} MANDATORY. A *xs:boolean*. This boolean indicates whether elements not cited in the {http location [p.47] } property MUST be appended to the request IRI or ignored. If the value of this property is "false", the rules defined in section **6.8.2.2.3 Serialization in**

the request IRI [p.60] dictate how to serialize elements not cited in {http location [p.47] } in the request IRI. Otherwise, those are NOT serialized in the request IRI.

The XML representation for this property is an *attribute information item* with the following Infoset properties:

- An OPTIONAL `ignoreUncited` *attribute information item* with the following Infoset properties:
    - A [local name] of `ignoreUncited`
    - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
    - A type of *xs:boolean*

The mapping from the XML representation to component properties is as follows:

Table 6-8. Mapping from XML Representation to Binding Operation component Extension Properties

Property	Value
{http location ignore uncited [p.59] }	The actual value of the `whttp:ignoreUncited` *attribute information item*, if present. Otherwise, "false".

6.8.2.2.3 Serialization in the request IRI

If the HTTP request method used does not allow HTTP message body (e.g. "GET" and "DELETE"), and if the value of the {http location ignore uncited [p.59] } property is "false", then the following rules apply.

If the {http location [p.47] } property is not present, or if it is present and its value does not contain a "?" (question mark) character, one is appended to the request IRI. If it does already contain a question mark character, then the value of the {http query parameter separator [p.48] } property, if present, or the value of the {http query parameter separator default [p.48] } property otherwise, is appended.

Finally, the query string computed in **6.8.2.2.1 Construction of the query string** [p.59] is appended.

Example 6-2. Instance data serialized in a IRI

The instance data defined in Example 6-1 [p.59] with the following `operation` declaration:

```
<operation ref='t:data'
    whttp:location='temperature/{town}'
    whttp:method='GET' />
```

and the following `endpoint` declaration:

```
<endpoint name='e' binding='t:b'
    address='http://ws.example.com/service1/' />
```

will serialize the message in the HTTP request as follows:

```
GET http://ws.example.com/service1/
    temperature/Fr%C3%A9jus?date=2006-01-06&unit=C HTTP/1.1
Host: ws.example.com
```

6.8.2.2.4 Serialization in the message body

If the HTTP request method used does allow an HTTP message body (e.g. "POST" and "PUT"), then the following rules apply.

Finally, the query string computed in **6.8.2.2.1 Construction of the query string** [p.59] is used as the value of the HTTP message body.

The `Content-Type` HTTP header field must have the value `application/x-www-form-urlencoded`.

Example 6-3. Instance data serialized in the HTTP Request IRI and message body

The instance data defined in Example 6-1 [p.59] with the following `operation` declaration:

```
<operation ref='t:data'
    whttp:inputSerialization='application/x-www-form-urlencoded'
    whttp:location='temperature/{town/}'
    whttp:method='POST' />
```

and the following `endpoint` declaration:

```
<endpoint name='e' binding='t:b'
    address='http://ws.example.com/service1/' />
```

will serialize the message in the HTTP request as follow:

```
POST http://ws.example.com/service1/temperature/Fr%C3%A9jus HTTP/1.1
Host: ws.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: ...

date=2006-01-06&unit=C
```

6.8.3 Serialization as "application/xml"

In this serialization, for HTTP requests, the rules for constructing the HTTP request IRI defined in **6.8.1 Serialization of the instance data in parts of the HTTP request IRI** [p.57] apply if the {style} property of the Interface Operation bound has a value of "http://www.w3.org/2006/01/wsdl/style/iri" as defined in **4.2 IRI Style** [p.20] .

The instance data [p.41] of the input, output or fault message is serialized as an XML document in the message body of the HTTP message, following the serialization defined in [*Canonical XML [p.68]*]. Therefore, it is only suitable for HTTP requests using methods allowing message bodies (i.e., for the HTTP binding defined in this specification, input messages where the HTTP method selected has a body), and for HTTP responses (i.e. output and fault messages for the HTTP binding defined in this specification).

The `Content-Type` HTTP header MUST have the value `application/xml`, or a media type compatible with `application/xml` as specified in section **6.3.2.1 Serialization rules for XML messages** [p.44] . Other HTTP headers, such as `Content-Encoding` or `Transfer-Encoding`, MAY be used.

6.8.4 Serialization as "multipart/form-data"

In this serialization, for HTTP requests, the rules for constructing the HTTP request IRI defined in **6.8.1 Serialization of the instance data in parts of the HTTP request IRI** [p.57] apply if the {style} property of the Interface Operation bound has a value of "http://www.w3.org/2006/01/wsdl/style/iri" as defined in **4.2 IRI Style** [p.20] .

This format is for legacy compatibility to permit the use of XForms clients with [*IETF RFC 2388 [p.68]*] servers. This serialization format may only be used when binding Interface Operation whose {style} property has a value of "http://www.w3.org/2006/01/wsdl/style/multipart" as defined in **4.3 Multipart style** [p.21] , i.e. this serialization format may only be used to serialize the HTTP request corresponding to the initial message of an interface operation.

Specifically, for the HTTP binding defined in this section (**6. WSDL HTTP Binding Extension** [p.41]), "multipart/form-data" MAY be used as a serialization format [p.43] for an input message (HTTP Request), but MUST NOT be used as a serialization format [p.43] for an output or fault message (HTTP Response). This format serializes the instance data in the HTTP message body, making it only suitable for HTTP requests using methods allowing message bodies.

Each element in the sequence is serialized into a part as follow:

1. The `Content-Disposition` header MUST have the value `form-data`, and its `name` parameter is the local name of the element.
2. The `Content-Type` header MUST have the value:
   - `application/xml` (or a media type compatible with `application/xml`) if the element has a complex type;
   - `application/octet-stream` if the element is of type `xs:base64Binary`, `xs:hexBinary`, or a derived type;
   - `text/plain` if the element has a simple type; The charset MUST be set appropriately. UTF-8 or UTF-16 MUST be at least supported.
3. If the type is `xs:base64Binary`, `xs:hexBinary`, `xs:anySimpleType` or a derived type, the content of the part is the content of the element. If the type is a complex type, the element is serialized following the rules defined in the **6.8.3 Serialization as application/xml** [p.61] .

It is an error for the instance data [p.41] to contain elements with an `xs:nil` attribute whose value is "true".

Example 6-4. Example of multipart/form-data

The following instance data of an input message:

```
<data>
  <town>
    <name>Fréjus</name>
    <country>France</country>
  </town>
  <date>2006-01-06</date>
</data>
```

with the following `operation` element

```
<operation ref='t:data'
    whttp:location='temperature'
    whttp:method='POST'
    whttp:inputSerialization='multipart/form-data'/>
```

will serialize the message as follow:

```
Content-Type: multipart/form-data; boundary=AaB03x
Content-Length: xxx

--AaB03x
Content-Disposition: form-data; name="town"
Content-Type: application/xml

<town>
  <name>Fréjus</name>
  <country>France</country>
</town>
--AaB03x
Content-Disposition: form-data; name="date"
Content-Type: text/plain; charset=utf-8

2006-01-06
--AaB03x--
```

6.9 Specifying the Transfer Coding

6.9.1 Description

Every Binding Message Reference and Interface Fault Reference component MAY indicate which transfer codings, as defined in section 3.6 of [*IETF RFC 2616 [p.68]*], are available for this particular message.

The HTTP binding extension provides a mechanism for indicating a default value at the Binding component and Binding Operation levels.

If no value is specified, no claim is being made.

6.9.2 Relationship to WSDL Component Model

The HTTP binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http transfer coding default} OPTIONAL. A *xs:string* to the Binding component. This property indicates the default transfer codings available for all Interface Message Reference and Interface Fault Reference components of any Interface component that uses this Binding component. Its value is ignored when the value of the {http version [p.46] } property is "1.0".
- {http transfer coding default} OPTIONAL. A *xs:string* to the Binding Operation component. This property indicates the default transfer codings available for all Binding Message Reference and Binding Fault components of this Binding Operation component. Its value is ignored when the value of the {http version [p.46] } property is "1.0".
- {http transfer coding} OPTIONAL. A *xs:string* to the Binding Message Reference component. This property indicates the transfer codings available for this Binding Message Reference component. Its value is ignored when the value of the {http version [p.46] } property is "1.0".
- Similarly, {http transfer coding} OPTIONAL, to the Binding Fault component

6.9.3 XML Representation

```
<description>
  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI"
           whttp:transferCodingDefault="xs:string"? >

    <fault ref="xs:QName"
           whttp:transferCoding="xs:string"? >
    </fault>*

    <operation location="xs:anyURI"?
               whttp:transferCodingDefault="xs:string"? >
      <input messageLabel="xs:NCName"?
             whttp:transferCoding="xs:string"? />

      <output messageLabel="xs:NCName"?
              whttp:transferCoding="xs:string"? />

    </operation>
  </binding>
</description>
```

The XML representation for specifying the transfer coding is an OPTIONAL *attribute information item* for the `input` , `output` , and `fault` *element information item*s with the following Infoset properties:

- A [local name] of `transferCoding`

- A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
- A type of *xs:string*

The XML representation for specifying the default transfer coding is an OPTIONAL *attribute information item* for the `binding` *element information item* or `binding` 's child `operation` *element information items* with the following Infoset properties:

- A [local name] of `transferCodingDefault`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
- A type of *xs:string*

6.9.4 Mapping from XML Representation to Component Properties

See Table 6-9 [p.65] .

Table 6-9. Mapping from XML Representation to Interface Message Reference component Extension Properties

Property	Value
{http transfer coding default [p.64] } of the Binding component	The actual value of the `whttp:transferCodingDefault` *attribute information item* of the `binding` *element information item*, if present.
{http transfer coding default [p.64] } of the Binding Operation component	The actual value of the `whttp:transferCodingDefault` *attribute information item* of the `operation` *element information item*, if present.
{http transfer coding [p.64] } of the Binding Message Reference component	The actual value of the `whttp:transferCoding` *attribute information item* of the `input` or `output` *element information item*, if present.
{http transfer coding [p.64] } of the Binding Fault component	The actual value of the `whttp:transferCoding` *attribute information item* of the `fault` *element information item*, if present.

6.10 Specifying the Use of HTTP Cookies

6.10.1 Description

Every Binding component MAY indicate whether HTTP cookies (as defined by [*IETF RFC 2965 [p.69]*]) are used for some or all of operations of the interface that this binding applies to.

6.10.2 Relationship to WSDL Component Model

The HTTP binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http cookies} REQUIRED. A *xs:boolean* to the Binding component.

6.10.3 XML Representation

```
<description>
  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI"
          whttp:cookies="xs:boolean"? >
  </binding>
</description>
```

The XML representation for specifying the use of HTTP cookies is an OPTIONAL *attribute information item* with the following Infoset properties:

- A [local name] of `cookies`
- A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
- A type of *xs:boolean*

6.10.4 Mapping from XML Representation to Component Properties

See Table 6-10 [p.66] .

Table 6-10. Mapping from XML Representation to Binding component Extension Properties

Property	Value
{http cookies [p.66] }	The actual value of the `whttp:cookies` *attribute information item*; otherwise, "false".

6.11 Specifying HTTP Access Authentication

6.11.1 Description

Every Endpoint component MAY indicate the use of an HTTP access authentication mechanism (as defined by [*IETF RFC 2616 [p.68]*]) for the endpoint described.

This binding extension specification allows the authentication scheme and realm to be specified.

6.11.2 Relationship to WSDL Component Model

The HTTP binding extension specification adds the following property to the WSDL component model (as defined in [*WSDL 2.0 Core Language [p.70]*]):

- {http authentication scheme} OPTIONAL. A *xs:token* with one of the values "basic" or "digest", to the Endpoint component, corresponding to the HTTP authentication scheme used. When present, this property indicates the authentication scheme in use: "basic" indicates the Basic Access Authentication scheme defined in [*IETF RFC 2617 [p.68]*], and "digest" indicates the Digest Access Authentication scheme as defined in [*IETF RFC 2617 [p.68]*].
- {http authentication realm} OPTIONAL. A *xs:string* to the Endpoint component. It corresponds to the realm authentication parameter defined in [*IETF RFC 2617 [p.68]*]. If the {http authentication scheme [p.67] } property is present, then this property MUST be present.

6.11.3 XML Representation

```
<description>
  <service>
    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? >
              whttp:authenticationType="xs:token"?
              whttp:authenticationRealm="xs:string"? />
    </endpoint>
  </service>
</description>
```

The XML representation for specifying the use of HTTP access authentication is two OPTIONAL *attribute information items* with the following Infoset properties:

- An OPTIONAL `authenticationType` *attribute information item* with the following Infoset properties:
  - A [local name] of `authenticationType`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:token* where the allowed token values are "basic" and "digest".
- An OPTIONAL `authenticationRealm` *attribute information item* with the following Infoset properties:
  - A [local name] of `authenticationRealm`
  - A [namespace name] of "http://www.w3.org/2006/01/wsdl/http"
  - A type of *xs:string*

6.11.4 Mapping from XML Representation to Component Properties

See Table 6-11 [p.68] .

Table 6-11. Mapping from XML Representation to Endpoint component Extension Properties

Property	Value
{http authentication scheme [p.67] }	The actual value of the `whttp:authenticationType` *attribute information item*, if present.
{http authentication realm [p.67] }	The actual value of the `whttp:authenticationRealm` *attribute information item*, if present; otherwise, if the `whttp:authenticationType` *attribute information item* is present, "" (the empty value).

6.12 Conformance

An *element information item* whose namespace name is "http://www.w3.org/2006/01/wsdl" and whose local part is `description` conforms to this binding extension specification if the *element information item*s and *attribute information item*s whose namespace is http://www.w3.org/2006/01/wsdl/http conform to the XML Schema for that element or attribute as defined by this specification and additionally adheres to all the constraints contained in this specification.

7. References

7.1 Normative References

[Canonical XML]
Canonical XML, J. Boyer, Author. World Wide Web Consortium, 15 March 2001. This version of the Canonical XML Recommendation is http://www.w3.org/TR/2001/REC-xml-c14n-20010315. The latest version of Canonical XML is available at http://www.w3.org/TR/xml-c14n.

[IETF RFC 2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2119.txt.

[IETF RFC 2388]
Returning Values from Forms: multipart/form-data, L. Masinter, Author. Internet Engineering Task Force, August 1998. Available at http://www.ietf.org/rfc/rfc2388.txt.

[IETF RFC 2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[IETF RFC 2617]
HTTP Authentication: Basic and Digest Access Authentication, J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[IETF RFC 2818]
HTTP Over TLS, E. Rescorla, Author. Internet Engineering Task Force, May 2000. Available at http://www.ietf.org/rfc/rfc2818.txt.

[IETF RFC 2965]
HTTP State Management Mechanism, D. Kristol, L. Montulli Authors. Internet Engineering Task Force, October 2000. Available at http://www.ietf.org/rfc/rfc2965.txt.

[IETF RFC 3023]
XML Media Types, M. Murata, S. St. Laurent, D. Kohn, Authors. Internet Engineering Task Force, January 2001. Available at http://www.ietf.org/rfc/rfc3023.txt.

[IETF RFC 3205]
On the use of HTTP as a Substrate, K. Moore, Authors. Internet Engineering Task Force, February 2002. Available at http://www.ietf.org/rfc/rfc3205.txt.

[IETF RFC 3986]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3986.txt.

[IETF RFC 3987]
Internationalized Resource Identifiers (IRIs), M. Duerst, M. Suignard, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3987.txt.

[XForms 1.0]
XForms 1.0, M. Dubinko, et al., Editors. World Wide Web Consortium, 14 October 2003. This version of the XForms 1.0 Recommendation is http://www.w3.org/TR/2003/REC-xforms-20031014/. The latest version of XForms 1.0 is available at http://www.w3.org/TR/xforms/.

[SOAP 1.2 Part 1: Messaging Framework]
SOAP Version 1.2 Part 1: Messaging Framework, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, Editors. World Wide Web Consortium, 24 June 2003. This version of the "SOAP Version 1.2 Part 1: Messaging Framework" Recommendation is http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. The latest version of "SOAP Version 1.2 Part 1: Messaging Framework" is available at http://www.w3.org/TR/soap12-part1/.

[SOAP 1.2 Part 2: Adjuncts]
SOAP Version 1.2 Part 2: Adjuncts, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H. Frystyk Nielsen, Editors. World Wide Web Consortium, 7 May 2003. This version of the "SOAP Version 1.2 Part 2: Adjuncts" Recommendation is http://www.w3.org/TR/2003/REC-soap12-part2-20030624/. The latest version of "SOAP Version 1.2 Part 2: Adjuncts" is available at http://www.w3.org/TR/soap12-part2/.

[Web Architecture]
Architecture of the World Wide Web, Volume One, I. Jacobs, and N. Walsh, Editors. World Wide Web Consortium, 15 December 2004. This version of the "Architecture of the World Wide Web, Volume One" Recommendation is http://www.w3.org/TR/2004/REC-webarch-20041215/. The latest version of "Architecture of the World Wide Web, Volume One" is available at http://www.w3.org/TR/webarch/.

[Web Services Architecture]
Web Services Architecture, David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, Editors. World Wide Web Consortium, 11 February 2004. This version of the "Web Services Architecture" Working Group Note is http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. The latest version of "Web Services Architecture" is available at http://www.w3.org/TR/ws-arch/.

[WSDL 2.0 Core Language]
Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, R. Chinnici, M. Gudgin, J-J. Moreau, S. Weerawarana, Editors. World Wide Web Consortium, 6 January 2006. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Specification is available is available at http://www.w3.org/TR/2006/CR-wsdl20-20060106. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" is available at http://www.w3.org/TR/wsdl20.

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Third Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Editors. World Wide Web Consortium, 4 February 2004. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2004/REC-xml-20040204/. The latest version of "Extensible Markup Language (XML) 1.0" is available at http://www.w3.org/TR/REC-xml.

[XML Information Set]
XML Information Set (Second Edition), J. Cowan and R. Tobin, Editors. World Wide Web Consortium, 4 February 2004. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2004/REC-xml-infoset-20040204. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset.

[XML Schema Structures]
XML Schema Part 1: Structures Second Edition, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 28 October 2004. This version of the XML Schema Part 1 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-1-20041028. The latest version of XML Schema Part 1 is available at http://www.w3.org/TR/xmlschema-1.

[XML Schema Datatypes]
XML Schema Part 2: Datatypes Second Edition, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 28 October 2004. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2.

7.2 Informative References

[SOAP Message Transmission Optimization Mechanism]
SOAP Message Transmission Optimization Mechanism, N. Mendelsohn, M. Nottingham, and H. Ruellan, Editors. World Wide Web Consortium, W3C Recommendation, 25 January 2005. This version of SOAP Message Transmission Optimization Mechanism is http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/. The latest version of the "SOAP Message Transmission Optimization Mechanism" document is available from http://www.w3.org/TR/soap12-mtom/.

[WSA 1.0 Core]
Web Services Addressing 1.0 - Core , M. Gudgin, M. Hadley, Editors. World Wide Web Consortium, 17 August 2005. This version of Web Services Addressing 1.0 - Core is http://www.w3.org/TR/2005/CR-ws-addr-core-20050817/ The latest version of the "Web Services Addressing 1.0 - Core" document is available from http://www.w3.org/TR/ws-addr-core.

[WSDL 2.0 Primer]
Web Services Description Language (WSDL) Version 2.0 Part 0: Primer , D.Booth, C.K. Liu , Editors. World Wide Web Consortium, 6 January 2006. This version of the "Web Services Descrip-

tion Language (WSDL) Version 2.0 Part 0: Primer" Specification is available at http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" is available at http://www.w3.org/TR/wsdl20-primer.

A. Acknowledgements (Non-Normative)

This document is the work of the W3C Web Service Description Working Group.

Members of the Working Group are (at the time of writing, and by alphabetical order): Charlton Barreto (Adobe Systems Inc.), Allen Brookes (Rogue Wave Softwave), Dave Chappell (Sonic Software), Helen Chen (Agfa-Gevaert N. V.), Roberto Chinnici (Sun Microsystems), Kendall Clark (University of Maryland), Glen Daniels (Sonic Software), Paul Downey (British Telecommunications), Youenn Fablet (Canon), Hugo Haas (W3C), Tom Jordahl (Macromedia), Anish Karmarkar (Oracle Corporation), Jacek Kopecky (DERI Innsbruck at the Leopold-Franzens-Universität Innsbruck, Austria), Amelia Lewis (TIBCO Software, Inc.), Michael Liddy (Education.au Ltd.), Kevin Canyang Liu (SAP AG), Jonathan Marsh (Microsoft Corporation), Josephine Micallef (SAIC - Telcordia Technologies), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Cyclone Commerce), Jean-Jacques Moreau (Canon), Mark Nottingham (BEA Systems, Inc.), David Orchard (BEA Systems, Inc.), Vivek Pandey (Sun Microsystems), Bijan Parsia (University of Maryland), Gilbert Pilz (BEA Systems, Inc.), Tony Rogers (Computer Associates), Arthur Ryman (IBM), Adi Sakala (IONA Technologies), Asir Vedamuthu (Microsoft Corporation), Sanjiva Weerawarana (WSO2), Ümit Yalçınalp (SAP AG).

Previous members were: Lily Liu (webMethods, Inc.), Don Wright (Lexmark), Joyce Yang (Oracle Corporation), Daniel Schutzer (Citigroup), Dave Solo (Citigroup), Stefano Pogliani (Sun Microsystems), William Stumbo (Xerox), Stephen White (SeeBeyond), Barbara Zengler (DaimlerChrysler Research and Technology), Tim Finin (University of Maryland), Laurent De Teneuille (L'Echangeur), Johan Pauhlsson (L'Echangeur), Mark Jones (AT&T), Steve Lind (AT&T), Sandra Swearingen (U.S. Department of Defense, U.S. Air Force), Philippe Le Hégaret (W3C), Jim Hendler (University of Maryland), Dietmar Gaertner (Software AG), Michael Champion (Software AG), Don Mullen (TIBCO Software, Inc.), Steve Graham (Global Grid Forum), Steve Tuecke (Global Grid Forum), Michael Mahan (Nokia), Bryan Thompson (Hicks & Associates), Ingo Melzer (DaimlerChrysler Research and Technology), Sandeep Kumar (Cisco Systems), Alan Davies (SeeBeyond), Jacek Kopecky (Systinet), Mike Ballantyne (Electronic Data Systems), Mike Davoren (W. W. Grainger), Dan Kulp (IONA Technologies), Mike McHugh (W. W. Grainger), Michael Mealling (Verisign), Waqar Sadiq (Electronic Data Systems), Yaron Goland (BEA Systems, Inc.), Ümit Yalçınalp (Oracle Corporation), Peter Madziak (Agfa-Gevaert N. V.), Jeffrey Schlimmer (Microsoft Corporation), Hao He (The Thomson Corporation), Erik Ackerman (Lexmark), Jerry Thrasher (Lexmark), Prasad Yendluri (webMethods, Inc.), William Vambenepe (Hewlett-Packard Company), David Booth (W3C), Sanjiva Weerawarana (IBM), Charlton Barreto (webMethods, Inc.), Asir Vedamuthu (webMethods, Inc.), Igor Sedukhin (Computer Associates), Martin Gudgin (Microsoft Corporation), Rebecca Bergersen (IONA Technologies), Ugo Corda (SeeBeyond).

The people who have contributed to discussions on www-ws-desc@w3.org are also gratefully acknowledged.

B. Component Summary (Non-Normative)

Table B-1 [p.72] lists all the components in the WSDL 2.0 Adjuncts abstract Component Model, and all their properties.

Table B-1. Summary of WSDL 2.0 Adjuncts Components and their Properties

Component	Defined Properties
Binding	{http transfer coding default [p.64] }, {http cookies [p.66] }, {http method default [p.47] }, {http query parameter separator default [p.48] }, {http version [p.46] }, {soap mep default [p.29] }, {soap modules [p.31] }, {soap underlying protocol [p.27] }, {soap version [p.26] }
Binding Fault	{http error status code [p.55] }, {http headers [p.52] }, {http transfer coding [p.64] }, {soap fault code [p.28] }, {soap fault subcodes [p.28] }, {soap headers [p.34] }, {soap modules [p.31] }
Binding Fault Reference	{soap modules [p.31] }
Binding Message Reference	{http headers [p.51] }, {http transfer coding [p.64] }, {soap headers [p.34] }, {soap modules [p.31] }
Binding Operation	{http location [p.47] }, {http transfer coding default [p.64] }, {http fault serialization [p.48] }, {http input serialization [p.48] }, {http location ignore uncited [p.59] }, {http method [p.48] }, {http output serialization [p.48] }, {http query parameter separator [p.48] }, {soap action [p.29] }, {soap mep [p.29] }, {soap modules [p.31] }
Endpoint	{http authentication realm [p.67] }, {http authentication scheme [p.67] }
HTTP Header [p.52]	{name [p.52] }, {parent [p.52] }, {required [p.52] }, {type definition [p.52] }
Interface Operation	{rpc signature [p.18] }, {safety [p.15] }
SOAP Header Block [p.34]	{element declaration [p.34] }, {mustUnderstand [p.34] }, {parent [p.35] }, {required [p.35] }
SOAP Module [p.31]	{parent [p.32] }, {ref [p.31] }, {required [p.32] }
Property	**Where Defined**
element declaration	SOAP Header Block.{element declaration [p.34] }
http authentication realm	Endpoint.{http authentication realm [p.67] }

http authentication scheme	Endpoint.{http authentication scheme [p.67] }
http cookies	Binding.{http cookies [p.66] }
http error status code	Binding Fault.{http error status code [p.55] }
http fault serialization	Binding Operation.{http fault serialization [p.48] }
http headers	Binding Fault.{http headers [p.52] }, Binding Message Reference.{http headers [p.51] }
http location ignore uncited	Binding Operation.{http location ignore uncited [p.59] }
http method	Binding Operation.{http method [p.48] }
http method default	Binding.{http method default [p.47] }
http output serialization	Binding Operation.{http output serialization [p.48] }
http query parameter separator	Binding Operation.{http query parameter separator [p.48] }
http transfer coding	Binding Fault.{http transfer coding [p.64] }, Binding Message Reference.{http transfer coding [p.64] }
http version	Binding.{http version [p.46] }
mustUnderstand	SOAP Header Block.{mustUnderstand [p.34] }
name	HTTP Header.{name [p.52] }
parent	HTTP Header.{parent [p.52] }, SOAP Header Block.{parent [p.35] }, SOAP Module.{parent [p.32] }
ref	SOAP Module.{ref [p.31] }
required	HTTP Header.{required [p.52] }, SOAP Header Block.{required [p.35] }, SOAP Module.{required [p.32] }
rpc signature	Interface Operation.{rpc signature [p.18] }
safety	Interface Operation.{safety [p.15] }
soap action	Binding Operation.{soap action [p.29] }
soap fault code	Binding Fault.{soap fault code [p.28] }
soap fault subcodes	Binding Fault.{soap fault subcodes [p.28] }

soap headers	Binding Fault.{soap headers [p.34] }, Binding Message Reference.{soap headers [p.34] }
soap mep	Binding Operation.{soap mep [p.29] }
soap modules	Binding.{soap modules [p.31] }, Binding Fault.{soap modules [p.31] }, Binding Fault Reference.{soap modules [p.31] }, Binding Message Reference.{soap modules [p.31] }, Binding Operation.{soap modules [p.31] }
soap underlying protocol	Binding.{soap underlying protocol [p.27] }
soap version	Binding.{soap version [p.26] }
type definition	HTTP Header.{type definition [p.52] }

C. Part 2 Change Log (Non-Normative)

Date	Author	Description
20051122	HH	LC359: moved transfer coding from binding fault ref to binding fault in XML representations
20051117	JJM	LC358: fixed formatting in some examples.
20051113	HH	LC359: moved transfer coding from binding fault ref to binding fault
20051111	HH	Added SOAP MEP / WSDL MEP mapping as per resolution
20051111	HH	LC333: implemented resolution to accommodate interfaceless bindings
20051111	HH	LC362: added URI to fault propagation rules
20051111	HH	LC337: added media type range
20051111	HH	LC305: added reference to BNF pseudo-schemas in Part 1
20051111	AGR	Added assertion tables. Added Fault Propagation Rule assertions.
20051110	HH	LC304: implemented proposal
20051110	HH	LC345: allowed POST as application/x-www-form-urlencoded and reorganized HTTP binding serializations
20051109	HH	LC301: specified that {soap action} is for the initial message of an operation
20051027	HH	LC339: added `required` attribute to wsoap:header and whttp:header
20051027	HH	LC340: clarified cardinality of headers
20051027	HH	LC331: if the {message content model} property is "#any" in the HTTP binding, then the payload MUST be any one XML element.

20051027	HH	LC330: operation styles mandate that the {message content model} of the operation's messages is "#element"
20051027	HH	LC329: we do now have default rules for binding faults
20051027	HH	LC327: made both HTTP authentication properties optional
20051027	HH	LC326: changed type of {http authentication scheme}
20051027	HH	LC315: fixed HTTP header serialization and IRI identification.
20051020	HH	LC319: implemented detailed resolution.
20051020	HH	LC342: fixed typos
20051020	HH	LC349: improved section 2's introduction
20051013	HH	LC334: removed HTTP error reason phrase
20051013	HH	Fixed mark-up for declaring {soap modules}, {soap headers} and {http headers}
20051013	HH	LC323: removed text on HTTP Accept headers.
20051013	HH	LC321: clarified {soap mep} error.
20051012	RRC	LC344(5): changed order of union member types in the schema for the wrpc:signature extension
20050923	HH	LC341: renamed {element} into {element declaration} and fixed typo
20050923	HH	LC318: reorganized default declarations in bindings
20050923	HH	LC320: added {parent} property to nested components
20050923	HH	LC317: clarified applicability of application/x-www-url-encoded and multipart/form-data
20050923	HH	LC314: completed introduction
20050923	HH	LC306: wsdlx declaration clarification.
20050923	HH	LC322: section 6.3 Default Binding Rules clarification.
20050923	HH	LC324: fixed queryParameterSeparatorDefault and queryParameterSeparator definitions.
20050923	HH	LC325: fixed typo in `transferCodingDefault` definition.
20050923	HH	LC313: made {soap action}, {http location}, {http error reason phrase}, {http transfer coding} properties optional; did not do {soap fault subcodes} because of LC319.
20050923	HH	LC312: fixed typo in Section 2. Predefined Message Exchange Patterns.

20050902	RRC	LC316: Added definition of wrpc namespace in section 1.1 and changed wording of reference to example 4-1 in section 4.1.
20050728	HH	LC76d: spelled out conflict between mustUnderstand use and schema definition; clarified mustUnderstand definition.
20050728	HH	Clarified {soap action} scope for SOAP 1.2 binding.
20050728	HH	LC76c: added security consideration section.
20050725	RRC	LC75f: allowed extension attributes on RPC-style input/output elements.
20050707	aal	Modified 2.2.2 per text supplied by Jean-Jacques.
20050616	AGR	Fixed component table.
20050616	JJM	Added markup to list all the components and properties used in Part 2 (although this currently [wrongly] shows those of Part 1).
20050616	JJM	Fixed wrong component names for properties. Renamed HTTP Header Block to HTTP Header.
20050614	RRC	LC76a: Added comment requested by reviewer.
20050615	JJM	Further pass at adding markup for properties. Fixed issues with entities preventing validation.
20050615	JJM	Added <propdef> and <prop> markup around properties.
20050614	JJM	Finished adding <comp> markup around components.
20050613	JJM	Started adding <comp> markup around components.
20050613	JJM	LC122: replaced "binding" by "binding extension" where appropriate.
20050613	JJM	LC98: {soap mep} only applies to SOAP 1.2.
20050613	RRC	LC74c: changed `documentation` element cardinality to zero or more.
20050606	HH	LC79 & LC102: added editors note about one-way MEP defaulting for SOAP 1.2
20050606	HH	LC130: `wsoap:code` is now optional, and aligned `whttp:code`
20050602	HH	LC75c: introduced wsdlx namespace, moved safety to Part 2.
20050527	HH	LC74a: switched to IRIs
20050527	HH	LC80: defined fragment identifiers for defined components as proposed
20050520	JJM	LC97: Fixed specifying default values throughout the spec. Resolved incoherencies along the way.
20050519	aal	added template to guide readers when defining new message exchange patterns.

20050512	HH	LC110: referenced RFC2616 for `whttp:version`
20050512	HH	LC77a: clarified namespace and local name serialization in `application/x-www-url-encoded` serialization
20050509	RRC	LC118: Added clarification to step 2 of the algorithm to compute the function signature for an operation that uses the wrpc:signature extension.
20050509	RRC	LC89a: Added conformance requirement for RPC style.
20050505	aal	LC52c: state that soap faults have no reasonable default.
20050505	aal	LC76a: allow extensions to override faults in rulesets; LC76b: define "propagate" in rulesets.
20050429	RRC	LC97: Made the setting of default values for properties more consistent.
20050429	RRC	LC75g: RPC should allows element wildcards
20050422	HH	LC75d: RPC style; same input and output elements need named type
20050420	JJM	Fixed typos in RPC section (part of LC78).
20050413	AV	LC76d: made changes to `wsoap:header` and `whttp:header` (removed required and changed default binding rules)
20050412	RRC	LC75h: added note on multiple return values in rpc style
20050415	HH	LC28: ignoring transfer coding for HTTP/1.0
20050408	HH	LC17: added order preservation in `application/x-www-url-encoded` serialization
20050408	HH	LC69a: added `whttp:queryParameterSeparator`
20050408	HH	LC47: added `whttp:reasonPhrase`
20050408	HH	LC76d: added `whttp:header`
20050408	HH	Added `wsoap:module` at the Binding Fault component model as per 2005-04-07 telcon
20050407	HH	LC7: fixed RPC style glitches
20050406	HH	LC76d: added `wsoap:header`
20050331	HH	LC106: URI and Multipart styles are placing restrictions on the initial message of the MEP
20050331	HH	LC111: added reference to section 8 of RFC3205 for use of HTTP error codes
20050321	HH	LC48b: added link between WSDL and SOAP 1.2 MEPs in predefined MEPs section

20050321	HH	LC74d: removed constraint on LocalPart of the output element in RPC style
20050321	HH	LC108: fixed typo and added missing {soap modules} XML mapping
20050321	HH	LC88: fixed typo
20050317	HH	LC61a: Incorporated RPC style
20050316	HH	LC61a: Merged the old part 2 and part 3 documents

C.1 WSDL 2.0 Extensions Change Log

Date	Author	Description
20050613	JJM	LC122: Replaced "binding" by "binding extension" where appropriate.
20050222	aal	Implement editorial changes for LC39, LC40, LC48c.
20050220	AGR	LC50: Adopt proposal for definition of "node", adding "Note:" before second sentence.
20041209	aal	add clarifying language for fault propagation, per LC54/76.
20040713	aal	implement editorial changes requested after review by GlenD, in application data feature and module.
20040713	aal	address issues 233 & 112 all at once, by increasing level of all divs, adding new intro div, adding new div to contain features, renaming spec. Lotsa changes, what fun.
20040713	aal	s/Label/Message Label/g and s/{label}/{message label}/g. issue 230.
20040713	aal	replace "fault generation" with "fault propagation" (in almost all cases; one case of "generate" remains to indicate that it ends an exchange). issue 234.
20040713	aal	add language to introduction describing relationship between these MEPs and the MEPs defined by SOAP 1.2 (issue 232). This replaces the language found two items down (issue 191).
20040713	aal	add (hereafter, simply 'patterns') to intro (issue 231).
20040610	aal	add language to introduction describing relationship between these MEPs and the MEPs defined by SOAP 1.2 (issue 191).
20040225	aal	add in-optional-out per minutes of 20 feb 2004 telecon
20040212	aal	change {messageReference} to {label} and "Message Reference component" to "Label component" per 20040212 teleconference
20040205	aal	change all 'A' and 'B' message labels into 'Out' or 'In', depending upon direction.
20040205	aal	s/message pattern/message exchange pattern/gi

20031204	jcs	Removed change marks; note that some were on div2 tag and did not show when transformed into HTML.
20031204	jcs	Per 4 Dec 2003 telecon, decided to rename 'Asynchronous Out-In' pattern to 'Output-Optional-Input'.
20031105	aal	Fix titles of added patterns. Move them to be in conjunction with similar patterns.
20031022	aal	Per action item from October 16 teleconference, added the three patterns using message-triggers-fault as published on the mailing list (robust-in-only, robust-out-only, asynch-out-in).
20031022	aal	Added internal linkage (using specref) from patterns to the fault rulesets which they use.
20031022	aal	Per 9 and 16 Oct 2003 teleconferences, marked in-multi-out and out-multi-in patterns deleted.
20031022	aal	Per 16 Oct 2003 teleconference, added a paragraph/sentence stating that generation of a fault terminates an exchange.
20031007	JCS	Per 2 Oct 2003 teleconference, changed "broadcast" to "multicast" in the introduction.
20030922	JCS	Per 22 Sep 2003 meeting in Palo Alto, CA, removed "Pattern Review" editorial note; added specific editorial notes for In-Multi-Out and Out-Multi-In.
20030911	RRC	Changed the "name" property of the message reference component to "messageReference".
20030904	JCS	Incorporated clarifications suggested by W3C\David Booth.
20030801	JCS	Per 30 July meeting, added recommendations from patterns task force.
20030612	AAL	Added fault generation rulesets and references to them from patterns.
20030313	MJG	Changed to Part 2 (from Part 3)
20030306	JCS	Proposed name for MEP7.
20030305	JCS	Per 4 Mar 03 meeting, renamed 'message exchange pattern' to 'message pattern' or 'pattern', added pattern for request-response, added ednote about review of patterns.
20030217	MJG	Fixed some issues with entities and validity errors WRT ulists
20030212	JCS	Initial draft

C.2 WSDL 2.0 Bindings Change Log

Date	Author	Description
20050310	JJM	Replaced <definitions> with <description>.
20050310	JJM	Fixed missing fault pseudo-schema.
20050301	RRC	LC55: enabled use of whttp:transferCoding on Binding Fault Reference components.
20050301	RRC	LC55: enabled use of wsoap:module on Binding Fault Reference components.
20050221	HH	LC48b: highlighted relationship between SOAP and WSDL MEPs
20050211	HH	LC49: added conformance section to each of the bindings
20050120	HH	LC75q: removed wsdls namespace and XML 1.1 reference; limiting to XML 1.0
20050120	HH	LC21: implemented resolution from 16 Dec 2004 WS Description WG telcon
20041209	HH	LC86: completed pseudo-schemas with missing F&P occurrences
20041209	HH	LC85: clarified mapping of messages in an operation to HTTP request/response
20041209	HH	LC30: removed instances of provider/requester agents and replaced them by HTTP server/client
20041209	HH	LC29d: clarified modification of default of SOAP serialization rules
20041208	AV	Introduced SOAP version independent WSDL SOAP Binding. Added two new sections, "Specifying the SOAP Version" and "SOAP 1.2 Binding". Plus, lots of shuffling.
20041027	HH	LC57 &LC58: fixed typos
20041027	HH	LC51
20041027	HH	LC45: {http location} may or may not be a template
20041027	HH	LC44: URL serialization expressed in terms of the component model
20041027	HH	LC29e: URL serialization: disallowing nil elements in certain cases; clarifying that empty elements are OK
20041001	HH	LC29g: switched 3.8 (serializations) and 3.9 (styles)
20041001	HH	LC29f: it is an error to have nil elements in an instance data for multipart/form-data
20041001	HH	LC29a & LC29c: indicated that there is no suitable default fault code
20041001	HH	LC15: moved {http location} under bulleted list in section 2
20040920	HH	LC36 & LC2: added wsdls:* and xs:* in SOAP binding

20040920	HH	LC32: fixed errors due to operation name restriction in serialization examples
20040920	HH	LC36: added wsdls:* and xs:* in HTTP binding
20040920	HH	LC37: corrected rules to set operation properties values in HTTP binding
20040920	HH	LC33: removed "default" in SOAP binding's HTTP method selection
20040920	HH	LC13: removed remaining mentions of HTTP Operation Component
20040920	HH	LC12: added whttp:location in SOAP XML summary
20040909	HH	LC10: fixed typo in example 3.3
20040909	HH	LC11: made default attributes consistent with the following form: `wbinding:fooDefault`
20040730	HH	Removed `property` on `wsoap:module` in pseudo-schema.
20040730	HH	Removed AD Feature HTTP serialization.
20040729	HH	Added AD Feature support in HTTP binding.
20040727	HH	Clarified interaction between SOAP binding and HTTP binding properties
20040727	HH	Renamed http prefix whttp
20040727	SW	Implemented Umit's proposal to mark MTOM as one optimization mechanism.
20040726	HH	Restricted URI style with regards to QNames and added trailing / in URL-encoded syntax
20040723	HH	Addressed issue 246: limited MEP to In-Out, In-Only and Robust In-Only
20040723	HH	Addressed issue 226.
20040723	HH	Addressed 249: major reorganization of the HTTP binding to be presented in a functional way like the SOAP binding rather than in a syntactical way.
20040722	SW	Moved SOAP binding syntax summary to the top per request. Also fixed the value of the binding/@type property in the pseudo-schema to show that its a SOAP binding.
20040722	HH	Added HTTP error code attribute on fault binding. Added relationship between instance data and properties in the component model. Addresses issue 166.
20040722	HH	Renamed SOAP protocol into underlying protocol.
20040721	HH	Set the {type} property of binding for HTTP binding.
20040721	HH	Fixes for issue 177.
20040720	HH	Cross-referenced Part 1 properties.

20040720	HH	Specified default serialization format for HTTP binding, as well as made clear how the defined serialization formats apply constraints on interface operation styles
20040705	JJM	Added note to indicate only one element per SOAP body.
20040702	SW	Corrected how the SOAP binding is indicated .. I had forgotten about binding/@type!
20040625	SW	Made pseudo-syntax consistent with part1
20040624	SW	Update the rest of the SOAP binding stuff and consistified everything.
20040624	SW	Cleaned up how SOAP modules were described. Added default SOAP MEP stuff.
20040623	SW	Added default binding rules about HTTP URI generation.
20040623	SW	Added default binding rules about SOAP MEP selection and HTTP Method selection.
20040623	SW	Fixed up soapaction default rules
20040623	SW	Allowed use of MTOM for payload serialization
20040623	SW	Fixed up the wsoap:protocol section
20040618	SW	Re-introduced AII and EII entity refs.
20040618	SW	Made soap:module compose with nearest-wins rule.
20040606	DO	Cleanup on http binding section - had missed some properties. completed removal of @separator
20040604	DO	Major rewrite of http binding. Moved to component model, added http properties, added input/output serialization, removed @separator, added self as editor
20040526	SW	Removed wsoap:address
20040526	SW	Editorial/small corrections per F2F decisions
20040526	SW	Made soap binding be mostly attribute based per F2F decision
20040519	SW	removed spurious fault element inside binding/operation/{in,out}put from syntax summary
20040519	SW	Put in wsoap:module at operation level in the syntax summary (was missing)
20040519	SW	Removed old SOAP binding text
20040519	SW	Removed wsoap:header
20040519	JJM	Added SOAP Address section
20040519	JJM	Added SOAP Operation section

20040519	JJM	Replace reference to "XML" by "XML1.0"
20040519	JJM	Added SOAP Fault section
20040519	JJM	Added SOAP Header section
20040519	JJM	Added SOAP Module section
20040516	SW	Finished writing up soap:binding
20040516	SW	Added myself as an editor.
20040514	SW	Added default binding rules.
20040514	SW	Commented out old totally out of date SOAP binding.
20040514	JJM	Rework the binding and module sections. Reindent to match the structure of the HTTP binding.
20040511	JJM	Updated SOAP binding pseudo-schema, according to telcon 20040506.
20040511	JJM	Updated SOAP binding introduction.
20040401	JJM	Fixed one remaining occurrence of "verb" (instead of "method").
20040326	JJM	Sanitized ednotes. Added new ednotes indicating the SOAP binding needs work and the HTTP binding is (mostly) OK.
20040326	JJM	Added Philippe’s note on URIPath, as per telcon 20040325.
20040305	JJM	Removed the archaic MIME binding, now superseded by the HTTP binding anyway.
20040305	JJM	Included Philippe’s changes to the HTTP binding.
20031103	JJM	Fix new non-normative SOAP binding pseudo-schema.
20031102	SW	Updated SOAP binding.
20031102	SW	Change 1.2 to 2.0 per WG decision to rename.
20030606	JJM	Replaced <kw/> by <b/>. Indicated that pseudo-schemas are not normative
20030604	JJM	Reformated pseudo-syntax elements to match Part 1 layout
20030529	JCS	Incorporated text to resolve Issue 6e
20030523	JJM	Commented out MIME binding example; this is primer stuff.
20030523	JJM	Added pseudo-syntax to all sections.
20030523	JJM	Started converting the fault and headerfault sections to component model.
20030523	JJM	Complete the Multipart and x-www-form-urlencoded sections.

20030523	JJM	Fixed typos in HTTP binding (in particular added NOT in some section headers).
20030522	JCS	Added rules for serializing HTTP response
20030522	JCS	Added cardinality to pseudo schema for HTTP binding
20030522	JCS	Changes @transport to @protocol for SOAP binding
20030522	JJM	Incorporated remaining text from Philippe into the HTTP binding.
20030522	JJM	Polished the HTTP binding, split into subsections, added double curly brace escape mechanism, removed pseudo-schema.
20030521	JCS	Added rules for @verbDefault/@verb and @location.
20030514	JJM	Start converting the HTTP binding to the component model. The next thing to do will be to remove http:urlReplacement, etc. and incorporate instead Philippe's text.
20030313	MJG	Changed to Part 3 (from Part 2)
20030117	JCS	Incorporated resolution for Issue 5 (@encodingStyle). Referenced (rather than in-lined XML Schema).
20030117	JJM	Various editorial fixes.
20030116	JCS	Updated pseudo and XML Schema.
20030116	JJM	Added propertyConstraint section.
20030116	JJM	Added soap:module section.
20030115	JCS	Incorporated resolutions for Issue 25 (drop @use and @encoding), Issue 51 (headers reference element/type), and attribute roll up into text and schema. Began reworking SOAP HTTP binding to use Infoset model. Removed informative appendices 'Notes on URIs' and example WSDL documents; expect them to appear in the primer. Updated SOAP 1.2 references to CR.
20030114	JJM	Removed ednote saying Part 2 is out of synch with Part 1.
20030111	JJM	Incorporated resolution for issue 17 (role AII).
20030109	JJM	Incorporated resolution for issue 4 (Namespaces).
20020702	JJM	Added summary to prefix table.
20020628	JJM	Added out-of-synch-with-Part2 and not-soap12-yet ednote.
20020621	JJM	Commented out the link to the previous version. There is no previous version for 1.2 right now.
20020621	JJM	Rewrote the Notation Conventions section.
20020621	JJM	Added reference to part 0 in introduction. Renumbered references.

20020621	JJM	Simplified abstract and introduction.
20020621	JJM	Obtain the list of WG members from a separate file.
20020621	JJM	Updated stylesheet and DTDs to latest XMLP stylesheet and DTDs.
20020621	JJM	Deleted placeholder for appendix C "Location of Extensibility Elements", since this is part 1 stuff and extensibility has been reworked anyway.
20020621	JJM	Corrected link to issues lists
20020621	JJM	Updated title from "WSDL" to "Web Services Description Language". Now refer to part 1 as "Web Services... Part 1: Framework
20020621	JJM	Added Jeffrey as an editor :-). Removed Gudge (now on Part 2) :-(
20020411	JJM	Fixed typos noticed by Kevin Liu
20020301	JJM	Converted the "Schemas" sections
20020301	JJM	Converted the "Wire WSDL examples" sections
20020301	JJM	Converted the "Notes on URIs" sections
20020301	JJM	Converted the "Notational Conventions" sections
20020301	JJM	Converted the "References" sections
20020301	JJM	Converted the "MIME Binding" section to XML
20020221	JJM	Converted the "HTTP Binding" section to XML
20020221	JJM	Added placeholders for the "Wire examples" and "Schema" sections
20020221	JJM	Converted the "SOAP Binding" section to XML
20020221	JJM	Added the Change Log
20020221	JJM	Added the Status section
20020221	JJM	Simplified the introduction; referred to Part1 for a longer introduction
20020221	JJM	Renamed to "Part 2: Bindings"
20020221	JJM	Created from http://www.w3.org/TR/2001/NOTE-wsdl-20010315