INTERNATIONAL STANDARD

ISO/IEC 29361

First edition
2008-06-15

Information technology — Web Services Interoperability — WS-I Basic Profile Version 1.1

Technologies de l'information — Interopérabilité des services du Web — Profil de base WS-I, version 1.1

Reference number
ISO/IEC 29361:2008(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29361 was prepared by the Web Services Interoperability Organization (WS-I) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Information technology — Web Services Interoperability — WS-I Basic Profile Version 1.1

1 Scope and introduction

1.1 Scope

This International Standard defines the WS-I Basic Profile 1.1 (hereafter, "Profile"), consisting of a set of non-proprietary Web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability.

Section 1 introduces the Profile, and explains its relationships to other profiles.

Section 2, "Profile Conformance", explains what it means to be conformant to the Profile.

Each subsequent section addresses a component of the Profile, and consists of two parts: an overview detailing the component specifications and their extensibility points, followed by subsections that address individual parts of the component specifications. Note that there is no relationship between the section numbers in this International Standard and those in the referenced specifications.

1.2 Relationships to Other Profiles

This Profile is derived from the Basic Profile 1.0 by incorporating any errata to date and separating out those requirements related to the serialization of envelopes and their representation in messages. Such requirements are now part of the Simple SOAP Binding Profile 1.0, identified with a separate conformance claim. This separation is made to facilitate composability of Basic Profile 1.1 with any profile that specifies envelope serialization, including the Simple SOAP Binding Profile 1.0 and the Attachments Profile 1.0. A combined claim of conformance to both the Basic Profile 1.1 and the Simple SOAP Binding Profile 1.0 is roughly equivalent to a claim of conformance to the Basic Profile 1.0 plus published errata.

This Profile, composed with the Simple SOAP Binding Profile 1.0 supercedes the Basic Profile 1.0. The Attachments Profile 1.0 adds support for SOAP with Attachments, and is intended to be used in combination with this Profile.

1.3 Changes from Basic Profile Version 1.0

This specification is derived from the Basic Profile Version 1.0, and incorporates published errata against that specification. The most notable changes are:

- MESSAGE conformance target - Some requirements that had a MESSAGE conformance target in BP1.0 now use a new target, ENVELOPE. This facilitates alternate serialisations of the message, such as that described in the Attachments Profile.
- SOAP Binding - Requirements relating to the SOAP binding's serialization of the message have been moved to the Simple SOAP Binding Profile to facilitate other serializations.

1.4 Guiding Principles

The Profile was developed according to a set of principles that, together, form the philosophy of the Profile, as it relates to bringing about interoperability. This section documents these guidelines.

No guarantee of interoperability
It is impossible to completely guarantee the interoperability of a particular service. However, the Profile does address the most common problems that implementation experience has revealed to date.

Application semantics
Although communication of application semantics can be facilitated by the technologies that comprise the Profile, assuring the common understanding of those semantics is not addressed by it.

Testability
When possible, the Profile makes statements that are testable. However, such testability is not required. Preferably, testing is achieved in a non-intrusive manner (e.g., examining artifacts "on the wire").

Strength of requirements
The Profile makes strong requirements (e.g., MUST, MUST NOT) wherever feasible; if there are legitimate cases where such a requirement cannot be met, conditional requirements (e.g., SHOULD, SHOULD NOT) are used. Optional and conditional requirements introduce ambiguity and mismatches between implementations.

Restriction vs. relaxation
When amplifying the requirements of referenced specifications, the Profile may restrict them, but does not relax them (e.g., change a MUST to a MAY).

Multiple mechanisms
If a referenced specification allows multiple mechanisms to be used interchangeably, the Profile selects those that are well-understood, widely implemented and useful. Extraneous or underspecified mechanisms and extensions introduce complexity and therefore reduce interoperability.

Future compatibility
When possible, the Profile aligns its requirements with in-progress revisions to the specifications it references. This aids implementers by enabling a

graceful transition, and assures that WS-I does not 'fork' from these efforts. When the Profile cannot address an issue in a specification it references, this information is communicated to the appropriate body to assure its consideration.

Compatibility with deployed services

Backwards compatibility with deployed Web services is not a goal for the Profile, but due consideration is given to it; the Profile does not introduce a change to the requirements of a referenced specification unless doing so addresses specific interoperability issues.

Focus on interoperability

Although there are potentially a number of inconsistencies and design flaws in the referenced specifications, the Profile only addresses those that affect interoperability.

Conformance targets

Where possible, the Profile places requirements on artifacts (e.g., WSDL descriptions, SOAP messages) rather than the producing or consuming software's behaviors or roles. Artifacts are concrete, making them easier to verify and therefore making conformance easier to understand and less error-prone.

Lower-layer interoperability

The Profile speaks to interoperability at the application layer; it assumes that interoperability of lower-layer protocols (e.g., TCP, IP, Ethernet) is adequate and well-understood. Similarly, statements about application-layer substrate protocols (e.g., SSL/TLS, HTTP) are only made when there is an issue affecting Web services specifically; WS-I does not attempt to assure the interoperability of these protocols as a whole. This assures that WS-I's expertise in and focus on Web services standards is used effectively.

1.5 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.

Normative statements of requirements in the Profile (i.e., those impacting conformance, as outlined in "Conformance Requirements") are presented in the following manner:

Rnnnn*Statement text here.*

where "nnnn" is replaced by a number that is unique among the requirements in the Profile , thereby forming a unique requirement identifier.

Requirement identifiers can be considered to be namespace qualified, in such a way as to be compatible with QNames from Namespaces in XML. If there is no explicit namespace prefix on a requirement's identifier (e.g., "R9999" as opposed to "bp10:R9999"), it should be interpreted as being in the namespace identified by the conformance URI of the document section it occurs in. If it is qualified, the prefix should be interpreted according to the namespace mappings in effect, as documented below.

Some requirements clarify the referenced specification(s), but do not place additional constraints upon implementations. For convenience, clarifications are annotated in the following manner: C

Some requirements are derived from ongoing standardization work on the referenced specification(s). For convenience, such forward-derived statements are annotated in the following manner: xxxx, where "xxxx" is an identifier for the specification (e.g., "WSDL20" for WSDL Version 2.0). Note that because such work was not complete when this document was published, the specification that the requirement is derived from may change; this information is included only as a convenience to implementers.

Extensibility points in underlying specifications (see "Conformance Scope") are presented in a similar manner:

Ennnn*Extensibility Point Name - Description*

where "nnnn" is replaced by a number that is unique among the extensibility points in the Profile. As with requirement statements, extensibility statements can be considered namespace-qualified.

This specification uses a number of namespace prefixes throughout; their associated URIs are listed below. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

- **soap** - "http://schemas.xmlsoap.org/soap/envelope/"
- **xsi** - "http://www.w3.org/2001/XMLSchema-instance"
- **xsd** - "http://www.w3.org/2001/XMLSchema"
- **soapenc** - "http://schemas.xmlsoap.org/soap/encoding/"
- **wsdl** - "http://schemas.xmlsoap.org/wsdl/"
- **soapbind** - "http://schemas.xmlsoap.org/wsdl/soap/"
- **uddi** - "urn:uddi-org:api_v2"

1.6 Profile Identification and Versioning

This document is identified by a name (in this case, Basic Profile) and a version number (here, 1.1). Together, they identify a particular *profile instance*.

Version numbers are composed of a major and minor portion, in the form "major.minor". They can be used to determine the precedence of a profile instance; a higher version number (considering both the major and minor components) indicates that an instance is more recent, and therefore supersedes earlier instances.

Instances of profiles with the same name (e.g., "Example Profile 1.1" and "Example Profile 5.0") address interoperability problems in the same general scope (although some developments may require the exact scope of a profile to change between instances).

One can also use this information to determine whether two instances of a profile are backwards-compatible; that is, whether one can assume that conformance to an earlier profile instance implies conformance to a later one. Profile instances with the same name and major version number (e.g., "Example Profile 1.0" and "Example Profile 1.1") MAY be considered compatible. Note that this does not imply anything about compatibility in the other direction; that is, one cannot assume that conformance with a later profile instance implies conformance to an earlier one.

2 Profile Conformance

Conformance to the Profile is defined by adherence to the set of *requirements* defined for a specific *target*, within the *scope* of the Profile. This section explains these terms and describes how conformance is defined and used.

2.1 Conformance Requirements

Requirements state the criteria for conformance to the Profile. They typically refer to an existing specification and embody refinements, amplifications, interpretations and clarifications to it in order to improve interoperability. All requirements in the Profile are considered normative, and those in the specifications it references that are in-scope (see "Conformance Scope") should likewise be considered normative. When requirements in the Profile and its referenced specifications contradict each other, the Profile 's requirements take precedence for purposes of Profile conformance.

Requirement levels, using RFC2119 language (e.g., MUST, MAY, SHOULD) indicate the nature of the requirement and its impact on conformance. Each requirement is individually identified (e.g., R9999) for convenience.

For example;

R9999 *WIDGETs SHOULD be round in shape.*

This requirement is identified by "R9999", applies to the target WIDGET (see below), and places a conditional requirement upon widgets; i.e., although this requirement must be met to maintain conformance in most cases, there are some situations where there may be valid reasons for it not being met (which are explained in the requirement itself, or in its accompanying text).

Each requirement statement contains exactly one requirement level keyword (e.g., "MUST") and one conformance target keyword (e.g., "MESSAGE"). The conformance target keyword appears in bold text (e.g. "**MESSAGE**"). Other conformance targets appearing in non-bold text are being used strictly for their definition and NOT as a conformance target. Additional text may be included to illuminate a requirement or group of requirements (e.g., rationale and examples); however, prose surrounding requirement statements must not be considered in determining conformance.

Definitions of terms in the Profile are considered authoritative for the purposes of determining conformance.

None of the requirements in the Profile, regardless of their conformance level, should be interpreted as limiting the ability of an otherwise conforming implementation to apply security countermeasures in response to a real or perceived threat (e.g., a denial of service attack).

2.2 Conformance Targets

Conformance targets identify what artifacts (e.g., SOAP message, WSDL description, UDDI registry data) or parties (e.g., SOAP processor, end user) requirements apply to.

This allows for the definition of conformance in different contexts, to assure unambiguous interpretation of the applicability of requirements, and to allow conformance testing of artifacts (e.g., SOAP messages and WSDL descriptions) and the behavior of various parties to a Web service (e.g., clients and service instances).

Requirements' conformance targets are physical artifacts wherever possible, to simplify testing and avoid ambiguity.

The following conformance targets are used in the Profile:

- **MESSAGE** - protocol elements that transport the ENVELOPE (e.g., SOAP/HTTP messages)
- **ENVELOPE** - the serialization of the soap:Envelope element and its content
- **DESCRIPTION** - descriptions of types, messages, interfaces and their concrete protocol and data format bindings, and the network access points associated with Web services (e.g., WSDL descriptions) (from Basic Profile 1.0)
- **INSTANCE** - software that implements a wsdl:port or a uddi:bindingTemplate (from Basic Profile 1.0)
- **CONSUMER** - software that invokes an INSTANCE (from Basic Profile 1.0)
- **SENDER** - software that generates a message according to the protocol(s) associated with it (from Basic Profile 1.0)
- **RECEIVER** - software that consumes a message according to the protocol(s) associated with it (e.g., SOAP processors) (from Basic Profile 1.0)
- **REGDATA** - registry elements that are involved in the registration and discovery of Web services (e.g. UDDI tModels) (from Basic Profile 1.0)

2.3 Conformance Scope

The scope of the Profile delineates the technologies that it addresses; in other words, the Profile only attempts to improve interoperability within its own scope. Generally, the Profile's scope is bounded by the specifications referenced by it.

The Profile's scope is further refined by extensibility points. Referenced specifications often provide extension mechanisms and unspecified or open-ended configuration parameters; when identified in the Profile as an extensibility point, such a mechanism or parameter is outside the scope of the Profile, and its use or non-use is not relevant to conformance.

Note that the Profile may still place requirements on the use of an extensibility point. Also, specific uses of extensibility points may be further restricted by other profiles, to improve interoperability when used in conjunction with the Profile.

Because the use of extensibility points may impair interoperability, their use should be negotiated or documented in some fashion by the parties to a Web service; for example, this could take the form of an out-of-band agreement.

The Profile's scope is defined by the referenced specifications in Appendix A, as refined by the extensibility points in Appendix B.

2.4 Claiming Conformance

Claims of conformance to the Profile can be made using the following mechanisms, as described in Conformance Claim Attachment Mechanisms, when the applicable Profile requirements associated with the listed targets have been met:

- **WSDL 1.1 Claim Attachment Mechanism for Web Services Instances** - MESSAGE DESCRIPTION INSTANCE RECEIVER
- **WSDL 1.1 Claim Attachment Mechanism for Description Constructs** - DESCRIPTION
- **UDDI Claim Attachment Mechanism for Web Services Instances** - MESSAGE DESCRIPTION INSTANCE RECEIVER
- **UDDI Claim Attachment Mechanism for Web Services Registrations** - REGDATA

The conformance claim URI for this Profile is "http://ws-i.org/profiles/basic/1.1".

3 Messaging

This section of the Profile incorporates the following specifications by reference, and defines extensibility points within them:

- Simple Object Access Protocol (SOAP) 1.1
  Extensibility points:
  - E0001 - Header blocks - Header blocks are the fundamental extensibility mechanism in SOAP.
  - E0002 - Processing order - The order of processing of a SOAP envelope's components (e.g., headers) is unspecified, and therefore may need to be negotiated out-of-band.
  - E0003 - Use of intermediaries - SOAP Intermediaries is an underspecified mechanism in SOAP 1.1, and their use may require

out-of-band negotiation. Their use may also necessitate careful consideration of where Profile conformance is measured.
    - E0004 - soap:actor values - Values of the soap:actor attribute, other than the special uri 'http://schemas.xmlsoap.org/soap/actor/next' , represent a private agreement between parties of the web service.
    - E0005 - Fault details - the contents of a Fault's detail element are not prescribed by SOAP 1.1.
    - E0006 - Envelope serialization - The Profile does not constrain some aspects of how the envelope is serialized into the message.
- RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
  Extensibility points:
    - E0007 - HTTP Authentication - HTTP authentication allows for extension schemes, arbitrary digest hash algorithms and parameters.
    - E0008 - Unspecified Header Fields - HTTP allows arbitrary headers to occur in messages.
    - E0009 - Expect-extensions - The Expect/Continue mechanism in HTTP allows for expect-extensions.
    - E0010 - Content-Encoding - The set of content-codings allowed by HTTP is open-ended and any besides 'gzip', 'compress', or 'deflate' are an extensibility point.
    - E0011 - Transfer-Encoding - The set of transfer-encodings allowed by HTTP is open-ended.
    - E0012 - Upgrade - HTTP allows a connection to change to an arbitrary protocol using the Upgrade header.
    - E0024 - Namespace Attributes - Namespace attributes on soap:Envelope and soap:Header elements
    - E0025 - Attributes on soap:Body elements - Neither namespaced nor local attributes are constrained by SOAP 1.1.
- RFC2965: HTTP State Management Mechanism

3.1 SOAP Envelopes

The following specifications (or sections thereof) are referred to in this section of the Profile :

- SOAP 1.1, Section 4

SOAP 1.1 defines a structure for composing messages, the envelope. The Profile mandates the use of that structure, and places the following constraints on its use:

3.1.1 SOAP Envelope Structure

R9980 *An* **ENVELOPE** *MUST conform to the structure specified in SOAP 1.1 Section 4, "SOAP Envelope" (subject to amendment by the Profile).*

R9981 *An* **ENVELOPE** *MUST have exactly zero or one child elements of the* `soap:Body` *element.*

While the combination of R2201 and R2210 (below) clearly imply that there may be at most one child element of the `soap:Body`, there is no explicit requirement in the Profile that articulates this constraint, leading to some confusion.

3.1.2 SOAP Envelope Namespace

SOAP 1.1 states that an envelope with a document element whose namespace name is other than "http://schemas.xmlsoap.org/soap/envelope/" should be discarded. The Profile requires that a fault be generated instead, to assure unambiguous operation.

R1015 *A* **RECEIVER** *MUST generate a fault if they encounter an envelope whose document element is not `soap:Envelope`.*

3.1.3 SOAP Body Namespace Qualification

The use of unqualified element names may cause naming conflicts, therefore qualified names must be used for the children of `soap:Body`.

R1014 *The children of the `soap:Body` element in an* **ENVELOPE** *MUST be namespace qualified.*

3.1.4 Disallowed Constructs

XML DTDs and PIs may introduce security vulnerabilities, processing overhead and semantic ambiguity when used in envelopes. As a result, certain XML constructs are disallowed by section 3 of SOAP 1.1.

Although published errata NE05 (see http://www.w3.org/XML/xml-names-19990114-errata) allows the namespace declaration xmlns:xml="http://www.w3.org/XML/1998/namespace" to appear, some older processors considered such a declaration to be an error. These requirements ensure that conformant artifacts have the broadest interoperability possible.

R1008 *An* **ENVELOPE** *MUST NOT contain a Document Type Declaration.* C

R1009 *An* **ENVELOPE** *MUST NOT contain Processing Instructions.* C

R1033 *An* **ENVELOPE** *SHOULD NOT contain the namespace declaration xmlns:xml="http://www.w3.org/XML/1998/namespace".* C

R1034 *A* **DESCRIPTION** *SHOULD NOT contain the namespace declaration xmlns:xml="http://www.w3.org/XML/1998/namespace".* C

3.1.5 SOAP Trailers

The interpretation of sibling elements following the `soap:Body` element is unclear. Therefore, such elements are disallowed.

R1011 *An* **ENVELOPE** *MUST NOT have any element children of `soap:Envelope` following the `soap:Body` element.*

This requirement clarifies a mismatch between the SOAP 1.1 specification and the SOAP 1.1 XML Schema.

For example,

INCORRECT:
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <soap:Body>
    <p:Process xmlns:p='http://example.org/Operations' />
  </soap:Body>
  <m:Data xmlns:m='http://example.org/information' >
  Here is some data with the message
  </m:Data>
</soap:Envelope>
```

CORRECT:
```
<soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <soap:Body>
    <p:Process xmlns:p='http://example.org/Operations' >
          <m:Data xmlns:m='http://example.org/information' >
  Here is some data with the message
      </m:Data>
    </p:Process>
  </soap:Body>
</soap:Envelope>
```

3.1.6 SOAP encodingStyle Attribute

The `soap:encodingStyle` attribute is used to indicate the use of a particular scheme in the encoding of data into XML. However, this introduces complexity, as this function can also be served by the use of XML Namespaces. As a result, the Profile prefers the use of literal, non-encoded XML.

R1005 *An* **ENVELOPE** *MUST NOT contain `soap:encodingStyle` attributes on any of the elements whose namespace name is "http://schemas.xmlsoap.org/soap/envelope/".*

R1006 *An* **ENVELOPE** *MUST NOT contain `soap:encodingStyle` attributes on any element that is a child of `soap:Body`.*

R1007 *An* **ENVELOPE** *described in an rpc-literal binding MUST NOT contain `soap:encodingStyle` attribute on any element that is a grandchild of `soap:Body`.*

3.1.7 SOAP mustUnderstand Attribute

The `soap:mustUnderstand` attribute has a restricted type of "xsd:boolean" that takes only "0" or "1". Therefore, only those two values are allowed.

R1013 *An* **ENVELOPE** *containing a `soap:mustUnderstand` attribute MUST only use the lexical forms "0" and "1".* C

3.1.8 xsi:type Attributes

In many cases, senders and receivers will share some form of type information related to the envelopes being exchanged.

R1017 *A* **RECEIVER** *MUST NOT mandate the use of the `xsi:type` attribute in envelopes except as required in order to indicate a derived type (see XML Schema Part 1: Structures, Section 2.6.1).*

3.1.9 SOAP1.1 attributes on SOAP1.1 elements

R1032 *The `soap:Envelope`, `soap:Header`,* and *`soap:Body` elements in an* **ENVELOPE** *MUST NOT have attributes in the namespace `"http://schemas.xmlsoap.org/soap/envelope/"`.*

3.2 SOAP Processing Model

The following specifications (or sections thereof) are referred to in this section of the Profile:

- SOAP 1.1, Section 2

SOAP 1.1 defines a model for the processing of envelopes. In particular, it defines rules for the processing of header blocks and the envelope body. It also defines rules related to generation of faults. The Profile places the following constraints on the processing model:

3.2.1 Mandatory Headers

SOAP 1.1's processing model is underspecified with respect to the processing of mandatory header blocks. Mandatory header blocks are those children of the `soap:Header` element bearing a `soap:mustUnderstand` attribute with a value of "1".

R1025 *A* **RECEIVER** *MUST handle envelopes in such a way that it appears that all checking of mandatory header blocks is performed before any actual processing.* SOAP12

This requirement guarantees that no undesirable side effects will occur as a result of noticing a mandatory header block after processing other parts of the message.

3.2.2 Generating mustUnderstand Faults

The Profile requires that receivers generate a fault when they encounter header blocks targeted at them, that they do not understand.

R1027 *A* **RECEIVER** *MUST generate a "soap:MustUnderstand" fault when an envelope contains a mandatory header block (i.e., one that has a `soap:mustUnderstand` attribute with the value "1") targeted at the receiver (via `soap:actor`) that the receiver does not understand.*SOAP12

3.2.3 SOAP Fault Processing

When a fault is generated, no further processing should be performed. In request-response exchanges, a fault message will be transmitted to the sender of the request, and some application level error will be flagged to the user.

Both SOAP and this Profile use the term 'generate' to denote the creation of a SOAP Fault. It is important to realize that generation of a Fault is distinct from its transmission, which in some cases is not required.

R1028 *When a fault is generated by a* **RECEIVER**, *further processing SHOULD NOT be performed on the SOAP envelope aside from that which is necessary to rollback, or compensate for, any effects of processing the envelope prior to the generation of the fault.* SOAP12

R1029 *Where the normal outcome of processing a SOAP envelope would have resulted in the transmission of a SOAP response, but rather a fault is generated instead, a* **RECEIVER** *MUST transmit a fault in place of the response.* SOAP12

R1030 *A* **RECEIVER** *that generates a fault SHOULD notify the end user that a fault has been generated when practical, by whatever means is deemed appropriate to the circumstance.* SOAP12

3.3 SOAP Faults

3.3.1 Identifying SOAP Faults

Some consumer implementations erroneously use only the HTTP status code to determine the presence of a Fault. Because there are situations where the Web infrastructure changes the HTTP status code, and for general reliability, the Profile requires that they examine the envelope. A Fault is an envelope that has a single child element of the `soap:Body` element, that element being a `soap:Fault` element.

R1107 *A* **RECEIVER** *MUST interpret a SOAP message as a Fault when the* `soap:Body` *of the message has a single* `soap:Fault` *child.*

3.3.2 SOAP Fault Structure

The Profile restricts the content of the `soap:Fault` element to those elements explicitly described in SOAP 1.1.

R1000 *When an* **ENVELOPE** *is a Fault, the* `soap:Fault` *element MUST NOT have element children other than* `faultcode`, `faultstring`, `faultactor` and `detail`.

For example,

```
INCORRECT:
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <faultcode>soap:Client</faultcode>
  <faultstring>Invalid message format</faultstring>
  <faultactor>http://example.org/someactor</faultactor>
  <detail>There were <b>lots</b> of elements in the message
      that I did not understand
  </detail>
  <m:Exception xmlns:m='http://example.org/faults/exceptions' >
    <m:ExceptionType>Severe</m:ExceptionType>
  </m:Exception>
</soap:Fault>
```

```
CORRECT:
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <faultcode>soap:Client</faultcode>
  <faultstring>Invalid message format</faultstring>
  <faultactor>http://example.org/someactor</faultactor>
  <detail>
     <m:msg xmlns:m='http://example.org/faults/exceptions'>
         There were <b>lots</b> of elements in
         the message that I did not understand
     </m:msg>
     <m:Exception xmlns:m='http://example.org/faults/exceptions'>
       <m:ExceptionType>Severe</m:ExceptionType>
     </m:Exception>
   </detail>
</soap:Fault>
```

3.3.3 SOAP Fault Namespace Qualification

The children of the `soap:Fault` element are local to that element, therefore namespace qualification is unnecessary.

R1001 *When an* **ENVELOPE** *is a Fault, the element children of the* `soap:Fault` *element MUST be unqualified.*

For example,

```
INCORRECT:
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <soap:faultcode>soap:Client</soap:faultcode>
  <soap:faultstring>Invalid message format</soap:faultstring>
  <soap:faultactor>http://example.org/someactor</soap:faultactor>
  <soap:detail>
      <m:msg xmlns:m='http://example.org/faults/exceptions'>
          There were <b>lots</b> of elements in the message that
          I did not understand
      </m:msg>
  </soap:detail>
</soap:Fault>
```

CORRECT:
```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
                        xmlns='' >
  <faultcode>soap:Client</faultcode>
  <faultstring>Invalid message format</faultstring>
  <faultactor>http://example.org/someactor</faultactor>
  <detail>
      <m:msg xmlns:m='http://example.org/faults/exceptions'>
          There were <b>lots</b> of elements in the message that
          I did not understand
      </m:msg>
  </detail>
</soap:Fault>
```

3.3.4 SOAP Fault Extensibility

For extensibility, additional attributes are allowed to appear on the `detail` element and additional elements are allowed to appear as children of the `detail` element.

R1002 *A* **RECEIVER** *MUST accept faults that have any number of elements, including zero, appearing as children of the `detail` element. Such children can be qualified or unqualified.*

R1003 *A* **RECEIVER** *MUST accept faults that have any number of qualified or unqualified attributes, including zero, appearing on the `detail` element. The namespace of qualified attributes can be anything other than "http://schemas.xmlsoap.org/soap/envelope/".*

3.3.5 SOAP Fault Language

Faultstrings are human-readable indications of the nature of a fault. As such, they may not be in a particular language, and therefore the `xml:lang` attribute can be used to indicate the language of the faultstring.

Note that this requirement conflicts with the schema for SOAP appearing at its namespace URL. A schema without conflicts can be found at "http://ws-i.org/profiles/basic/1.1/soap-envelope-2004-01-21.xsd".

R1016 *A* **RECEIVER** *MUST accept faults that carry an `xml:lang` attribute on the `faultstring` element.*

3.3.6 SOAP Custom Fault Codes

SOAP 1.1 allows custom fault codes to appear inside the `faultcode` element, through the use of the "dot" notation.

Use of this mechanism to extend the meaning of the SOAP 1.1-defined fault codes can lead to namespace collision. Therefore, its use should be avoided, as doing so may cause interoperability issues when the same names are used in the right-hand side of the "." (dot) to convey different meaning.

Instead, the Profile encourages the use of the fault codes defined in SOAP 1.1, along with additional information in the `detail` element to convey the nature of the fault.

Alternatively, it is acceptable to define custom fault codes in a namespace controlled by the specifying authority.

A number of specifications have already defined custom fault codes using the "." (dot) notation. Despite this, their use in future specifications is discouraged.

R1004 *When an* **ENVELOPE** *contains a* `faultcode` *element, the content of that element SHOULD be either one of the fault codes defined in SOAP 1.1 (supplying additional information if necessary in the* `detail` *element), or a Qname whose namespace is controlled by the fault's specifying authority (in that order of preference).*

R1031 *When an* **ENVELOPE** *contains a* `faultcode` *element the content of that element SHOULD NOT use of the SOAP 1.1 "dot" notation to refine the meaning of the fault.*

It is recommended that applications that require custom fault codes either use the SOAP1.1 defined fault codes and supply additional information in the detail element, or that they define these codes in a namespace that is controlled by the specifying authority.

For example,

INCORRECT:

```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
            xmlns:c='http://example.org/faultcodes' >
  <faultcode>soap:Server.ProcessingError</faultcode>
  <faultstring>An error occurred while processing the message
  </faultstring>
</soap:Fault>
```

CORRECT:

```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
            xmlns:c='http://example.org/faultcodes' >
  <faultcode>c:ProcessingError</faultcode>
  <faultstring>An error occured while processing the message
  </faultstring>
</soap:Fault>
```

CORRECT:

```
<soap:Fault xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/' >
  <faultcode>soap:Server</faultcode>
  <faultstring>An error occured while processing the message
  </faultstring>
</soap:Fault>
```

3.4 Use of SOAP in HTTP

The following specifications (or sections thereof) are referred to in this section of the Profile:

- SOAP 1.1, Section 6
- HTTP/1.1
- HTTP State Management Mechanism

SOAP 1.1 defines a single protocol binding, for HTTP. The Profile mandates the use of that binding, and places the following constraints on its use:

3.4.1 HTTP Protocol Binding

Several versions of HTTP are defined. HTTP/1.1 has performance advantages, and is more clearly specified than HTTP/1.0.

R1141 *A* **MESSAGE** *MUST be sent using either HTTP/1.1 or HTTP/1.0.*

R1140 *A* **MESSAGE** *SHOULD be sent using HTTP/1.1.*

Note that support for HTTP/1.0 is implied in HTTP/1.1, and that intermediaries may change the version of a message; for more information about HTTP versioning, see RFC2145, "Use and Interpretation of HTTP Version Numbers."

3.4.2 HTTP Methods and Extensions

The SOAP1.1 specification defined its HTTP binding such that two possible methods could be used, the HTTP POST method and the HTTP Extension Framework's M-POST method. The Profile requires that only the HTTP POST method be used and precludes use of the HTTP Extension Framework.

R1132 *A HTTP request* **MESSAGE** *MUST use the HTTP POST method.*

R1108 *A* **MESSAGE** *MUST NOT use the HTTP Extension Framework (RFC2774).*

The HTTP Extension Framework is an experimental mechanism for extending HTTP in a modular fashion. Because it is not deployed widely and also because its benefits to the use of SOAP are questionable, the Profile does not allow its use.

3.4.3 SOAPAction HTTP Header

Testing has demonstrated that requiring the `SOAPAction` HTTP header field-value to be quoted increases interoperability of implementations. Even though HTTP allows unquoted header field-values, some SOAP implementations require that they be quoted.

`SOAPAction` is purely a hint to processors. All vital information regarding the intent of a message is carried in `soap:Envelope`.

R1109 *The value of the `SOAPAction` HTTP header field in a HTTP request* **MESSAGE** *MUST be a quoted string.* C

R1119 *A* **RECEIVER** *MAY respond with a fault if the value of the `SOAPAction` HTTP header field in a message is not quoted.* C

R1127 *A* **RECEIVER** *MUST NOT rely on the value of the `SOAPAction` HTTP header to correctly process the message.*SOAP12

For example,

CORRECT:

A WSDL Description that has:

```
<soapbind:operation soapAction="foo" />
```

results in a message with a SOAPAction HTTP header field of:

```
SOAPAction: "foo"
```

CORRECT:

A WSDL Description that has:

```
<soapbind:operation />
```

or

```
<soapbind:operation soapAction="" />
```

results in a message with a corresponding SOAPAction HTTP header field as follows:

```
SOAPAction: ""
```

3.4.4 HTTP Success Status Codes

HTTP uses the 2xx series of status codes to communicate success. In particular, 200 is the default for successful messages, but 202 can be used to indicate that a message has been submitted for processing. Additionally, other 2xx status codes may be appropriate, depending on the nature of the HTTP interaction.

R1124 *An* **INSTANCE** *MUST use a 2xx HTTP status code on a response message that indicates the successful outcome of a HTTP request.*

R1111 *An* **INSTANCE** *SHOULD use a "200 OK" HTTP status code on a response message that contains an envelope that is not a fault.*

R1112 *An* **INSTANCE** *SHOULD use either a "200 OK" or "202 Accepted" HTTP status code for a response message that does not contain a SOAP envelope but indicates the successful outcome of a HTTP request.*

Despite the fact that the HTTP 1.1 assigns different meanings to response status codes "200" and "202", in the context of the Profile they should be considered equivalent by the initiator of the request. The Profile accepts both status codes because some SOAP implementations have little control over the HTTP protocol implementation and cannot control which of these response status codes is sent.

3.4.5 HTTP Redirect Status Codes

There are interoperability problems with using many of the HTTP redirect status codes, generally surrounding whether to use the original method, or GET. The Profile mandates "307 Temporary Redirect", which has the semantic of redirection with the same HTTP method, as the correct status code for redirection. For more information, see the 3xx status code descriptions in RFC2616.

R1130 *An* **INSTANCE** *MUST use the "307 Temporary Redirect" HTTP status code when redirecting a request to a different endpoint.*

R1131 *A* **CONSUMER** *MAY automatically redirect a request when it encounters a "307 Temporary Redirect" HTTP status code in a response.*

RFC2616 notes that user-agents should not automatically redirect requests; however, this requirement was aimed at browsers, not automated processes (which many Web services will be). Therefore, the Profile allows, but does not require, consumers to automatically follow redirections.

3.4.6 HTTP Client Error Status Codes

HTTP uses the 4xx series of status codes to indicate failure due to a client error. Although there are a number of situations that may result in one of these codes, the Profile highlights those when the HTTP request does not have the proper media type, and when the anticipated method ("POST") is not used.

R1125 *An* **INSTANCE** *MUST use a 4xx HTTP status code for a response that indicates a problem with the format of a request.*

R1113 *An* **INSTANCE** *SHOULD use a "400 Bad Request" HTTP status code, if a HTTP request message is malformed.*

R1114 *An* **INSTANCE** *SHOULD use a "405 Method not Allowed" HTTP status code if a HTTP request message's method is not "POST".*

R1115 *An* **INSTANCE** *SHOULD use a "415 Unsupported Media Type" HTTP status code if a HTTP request message's Content-Type header field-value is not permitted by its WSDL description.*

Note that these requirements do not force an instance to respond to requests. In some cases, such as Denial of Service attacks, an instance may choose to ignore requests.

Also note that SOAP 1.1, Section 6.2 requires that SOAP Fault can only be returned with HTTP 500 "Internal Server Error" code. This profile doesn't change that requirement. When HTTP 4xx error status code is used, the response message should not contain a SOAP Fault.

3.4.7 HTTP Server Error Status Codes

HTTP uses the 5xx series of status codes to indicate failure due to a server error.

R1126 *An* **INSTANCE** *MUST return a "500 Internal Server Error" HTTP status code if the response envelope is a Fault.*

3.4.8 HTTP Cookies

The HTTP State Management Mechanism ("Cookies") allows the creation of stateful sessions between Web browsers and servers. Being designed for hypertext browsing, Cookies do not have well-defined semantics for Web services, and, because they are external to the envelope, are not accommodated by either SOAP 1.1 or WSDL 1.1. However, there are situations where it may be necessary to use Cookies; e.g., for load balancing between servers, or for integration with legacy systems that use Cookies. For these reasons, the Profile limits the ways in which Cookies can be used, without completely disallowing them.

R1120 *An* **INSTANCE** *MAY use the HTTP state mechanism ("Cookies").*

R1122 *An* **INSTANCE** *using Cookies SHOULD conform to RFC2965.*

R1121 *An* **INSTANCE** *SHOULD NOT require consumer support for Cookies in order to function correctly.*

R1123 *The value of the cookie MUST be considered to be opaque by the* **CONSUMER**.

The Profile recommends that cookies not be required by instances for proper operation; they should be a hint, to be used for optimization, without materially affecting the execution of the Web service. However, they may be required in legacy integration and other exceptional use cases, so requiring them does not make an instance non-conformant. While Cookies thus may have meaning to the instance, they should not be used as an out-of-bound data channel between the instance and the consumer. Therefore, interpretation of Cookies is not allowed at all by the consumer - it is required to treat them as opaque (i.e., have no meaning to the consumer).

4 Service Description

The Profile uses Web Services Description Language (WSDL) to enable the description of services as sets of endpoints operating on messages.

This section of the Profile incorporates the following specifications by reference, and defines extensibility points within them:

- Extensible Markup Language (XML) 1.0 (Second Edition)
- Namespaces in XML 1.0
- XML Schema Part 1: Structures
  Extensibility points:
  - E0017 - Schema annotations - XML Schema allows for annotations, which may be used to convey additional information about data structures.
- XML Schema Part 2: Datatypes
- Web Services Description Language (WSDL) 1.1
  Extensibility points:
  - E0013 - WSDL extensions - WSDL allows extension elements and attributes in certain places; use of such extensions requires out-of-band negotiation.
  - E0014 - Validation mode - whether the parser used to read WSDL and XML Schema documents performs DTD validation or not.

- E0015 - Fetching of external resources - whether the parser used to read WSDL and XML Schema documents fetches external entities and DTDs.
- E0016 - Relative URIs - WSDL does not adequately specify the use of relative URIs for the following: soapbind:body/@namespace, soapbind:address/@location, wsdl:import/@location, xsd:schema/@targetNamespace and xsd:import/@schemaLocation. Their use may require further coordination; see XML Base for more information.

4.1 Required Description

An instance of a Web service is required to make the contract that it operates under available in some fashion.

R0001 *Either an* **INSTANCE***'s WSDL 1.1 description, its UDDI binding template, or both MUST be available to an authorized consumer upon request.*

This means that if an authorized consumer requests a service description of a conformant service instance, then the service instance provider must make the WSDL document, the UDDI binding template, or both available to that consumer. A service instance may provide run-time access to WSDL documents from a server, but is not required to do so in order to be considered conformant. Similarly, a service instance provider may register the instance provider in a UDDI registry, but is not required to do so to be considered conformant. In all of these scenarios, the WSDL contract must exist, but might be made available through a variety of mechanisms, depending on the circumstances.

4.2 Document Structure

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 2.1

WSDL 1.1 defines an XML-based structure for describing Web services. The Profile mandates the use of that structure, and places the following constraints on its use:

4.2.1 WSDL Schema Definitions

The normative schemas for WSDL appearing in Appendix 4 of the WSDL 1.1 specification have inconsistencies with the normative text of the specification. The Profile references new schema documents that have incorporated fixes for known errors.

R2028 *A* **DESCRIPTION** *using the WSDL namespace (prefixed "wsdl" in this Profile) MUST be valid according to the XML Schema found at "http://ws-i.org/profiles/basic/1.1/wsdl-2004-08-24.xsd".*

R2029 *A* **DESCRIPTION** *using the WSDL SOAP binding namespace (prefixed "soapbind" in this Profile) MUST be valid according to the XML Schema found at "http://ws-i.org/profiles/basic/1.1/wsdlsoap-2004-08-24.xsd".*

Although the Profile requires WSDL descriptions to be Schema valid, it does not require consumers to validate WSDL documents. It is the responsibility of a WSDL document's author to assure that it is Schema valid.

4.2.2 WSDL and Schema Import

Some examples in WSDL 1.1 incorrectly show the WSDL import statement being used to import XML Schema definitions. The Profile clarifies use of the import mechanisms to keep them consistent and confined to their respective domains. Imported schema documents are also constrained by XML version and encoding requirements consistent to those of the importing WSDL documents.

R2001 *A* **DESCRIPTION** *MUST only use the WSDL "import" statement to import another WSDL description.*

R2803 *In a* **DESCRIPTION**, *the namespace attribute of the* `wsdl:import` *MUST NOT be a relative URI.*

R2002 *To import XML Schema Definitions, a* **DESCRIPTION** *MUST use the XML Schema "import" statement.*

R2003 *A* **DESCRIPTION** *MUST use the XML Schema "import" statement only within the* `xsd:schema` *element of the types section.*

R2004 *In a* **DESCRIPTION** *the schemaLocation attribute of an xsd:import element MUST NOT resolve to any document whose root element is not "schema" from the namespace "http://www.w3.org/2001/XMLSchema".*

R2009 *An XML Schema directly or indirectly imported by a* **DESCRIPTION** *MAY include the Unicode Byte Order Mark (BOM).*

R2010 *An XML Schema directly or indirectly imported by a* **DESCRIPTION** *MUST use either UTF-8 or UTF-16 encoding.*

R2011 *An XML Schema directly or indirectly imported by a* **DESCRIPTION** *MUST use version 1.0 of the eXtensible Markup Language W3C Recommendation.*

For example,

INCORRECT:
```
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote/definitions"
   xmlns:xsd1="http://example.com/stockquote/schemas"
                ...
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/schemas"
           location="http://example.com/stockquote/stockquote.xsd"/>
   <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
   </message>
                ...
</definitions>
```

CORRECT:
```
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote/definitions"
                   ...
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/definitions"
           location="http://example.com/stockquote/stockquote.wsdl"/>

   <message name="GetLastTradePriceInput">
      <part name="body" element="..."/>
   </message>
                   ...
   </definitions>
```

CORRECT:
```
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote/"
   xmlns:xsd1="http://example.com/stockquote/schemas"
                ...
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/definitions"
        location="http://example.com/stockquote/stockquote.wsdl"/>

   <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>
                ...
</definitions>
```

4.2.3 WSDL Import location Attribute Structure

WSDL 1.1 is not clear about whether the `location` attribute of the `wsdl:import` statement is required, or what its content is required to be.

> R2007 *A* **DESCRIPTION** *MUST specify a non-empty* `location` *attribute on the* `wsdl:import` *element.*

Although the `wsdl:import` statement is modeled after the `xsd:import` statement, the `location` attribute is required by `wsdl:import` while the corresponding attribute on `xsd:import`, `schemaLocation` is optional. Consistent with `location` being required, its content is not intended to be empty.

4.2.4 WSDL Import location Attribute Semantics

WSDL 1.1 is unclear about whether WSDL processors must actually retrieve and process the WSDL document from the URI specified in the `location` attribute on the `wsdl:import` statements it encounters.

> R2008 *A CONSUMER MAY, but need not, retrieve a WSDL description from the URI specified in the location attribute on a* `wsdl:import` *element.* C

The value of the location attribute of a `wsdl:import` element is a hint. A WSDL processor may have other ways of locating a WSDL description for a given namespace.

4.2.5 Placement of WSDL import Elements

Example 3 in WSDL 1.1 Section 3.1 causes confusion regarding the placement of `wsdl:import`.

R2022 *When they appear in a* **DESCRIPTION**, *`wsdl:import` elements MUST precede all other elements from the WSDL namespace except `wsdl:documentation`.*

R2023 *When they appear in a* **DESCRIPTION**, *`wsdl:types` elements MUST precede all other elements from the WSDL namespace except `wsdl:documentation` and `wsdl:import`.*

For example,

INCORRECT:

```
<definitions name="StockQuote"
           ...
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/definitions"
         location="http://example.com/stockquote/stockquote.wsdl"/>

   <message name="GetLastTradePriceInput">
       <part name="body" type="tns:TradePriceRequest"/>
   </message>
            ...
   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteSoap">
           ....
      </port>
   </service>

   <types>
      <schema targetNamespace="http://example.com/stockquote/schemas"
              xmlns="http://www.w3.org/2001/XMLSchema">
           .......
      </schema>
   </types>
</definitions>
```

CORRECT:

```
   <definitions name="StockQuote"
      targetNamespace="http://example.com/stockquote/definitions">

     <import namespace="http://example.com/stockquote/base"
       location="http://example.com/stockquote/stockquote.wsdl"/>

      <message name="GetLastTradePriceInput">
         <part name="body" element="..."/>
      </message>
                  ...
   </definitions>
```

CORRECT:

```
<definitions name="StockQuote"
           ...
```

```
   xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
     <schema targetNamespace="http://example.com/stockquote/schemas"
          xmlns="http://www.w3.org/2001/XMLSchema">
           .......
     </schema>
   </types>

   <message name="GetLastTradePriceInput">
        <part name="body" element="tns:TradePriceRequest"/>
   </message>
               ...
   <service name="StockQuoteService">
      <port name="StockQuotePort" binding="tns:StockQuoteSoap">
            ....
      </port>
   </service>
</definitions>
```

4.2.6 XML Version Requirements

Neither WSDL 1.1 nor XML Schema 1.0 mandate a particular version of XML. For interoperability, WSDL documents and the schemas they import expressed in XML must use version 1.0.

R4004 *A* **DESCRIPTION** *MUST use version 1.0 of the eXtensible Markup Language W3C Recommendation.*

4.2.7 XML Namespace declarations

Although published errata NE05 (see http://www.w3.org/XML/xml-names-19990114-errata) allows this namespace declaration to appear, some older processors considered such a declaration to be an error. This requirement ensures that conformant artifacts have the broadest interoperability possible.

R4005 *A DESCRIPTION SHOULD NOT contain the namespace declaration xmlns:xml="http://www.w3.org/XML/1998/namespace".*c

4.2.8 WSDL and the Unicode BOM

XML 1.0 allows documents that use the UTF-8 character encoding to include a BOM; therefore, description processors must be prepared to accept them.

R4002 *A* **DESCRIPTION** *MAY include the Unicode Byte Order Mark (BOM).*c

4.2.9 Acceptable WSDL Character Encodings

The Profile consistently requires either UTF-8 or UTF-16 encoding for both SOAP and WSDL.

R4003 *A* **DESCRIPTION** *MUST use either UTF-8 or UTF-16 encoding.*

4.2.10 Namespace Coercion

Namespace coercion on `wsdl:import` is disallowed by the Profile.

R2005 *The `targetNamespace` attribute on the `wsdl:definitions` element of a description that is being imported MUST have same the value as the `namespace` attribute on the `wsdl:import` element in the importing* **DESCRIPTION**.

4.2.11 WSDL documentation Element

The WSDL 1.1 schema and the WSDL 1.1 specification are inconsistent with respect to where `wsdl:documentation` elements may be placed.

R2030 *In a* **DESCRIPTION** *the wsdl:documentation element MAY be present as the first child element of `wsdl:import`, `wsdl:part` and `wsdl:definitions` in addition to the elements cited in the WSDL1.1 specification.* WSDL20

4.2.12 WSDL Extensions

Requiring support for WSDL extensions that are not explicitly specified by this or another WS-I Profile can lead to interoperability problems with development tools that have not been instrumented to understand those extensions.

R2025 *A* **DESCRIPTION** *containing WSDL extensions MUST NOT use them to contradict other requirements of the Profile.*

R2026 *A* **DESCRIPTION** *SHOULD NOT include extension elements with a `wsdl:required` attribute value of "true" on any WSDL construct (`wsdl:binding`, `wsdl:portType`, `wsdl:message`, `wsdl:types` or `wsdl:import`) that claims conformance to the Profile.*

R2027 *If during the processing of a description, a consumer encounters a WSDL extension element that has a `wsdl:required` attribute with a boolean value of "true" that the consumer does not understand or cannot process, the* **CONSUMER** *MUST fail processing.*

Development tools that consume a WSDL description and generate software for a Web service instance might not have built-in understanding of an unknown WSDL extension. Hence, use of required WSDL extensions should be avoided. Use of a required WSDL extension that does not have an available specification for its use and semantics imposes potentially insurmountable interoperability concerns for all but the author of the extension. Use of a required WSDL extension that has an available specification for its use and semantics reduces, but does not eliminate the interoperability concerns that lead to this refinement.

For the purposes of the Profile, all elements in the "http://schemas.xmlsoap.org/wsdl/" namespace are extensible via element as well as attributes. As a convenience, WS-I has published a version of the WSDL1.1 schema that reflects this capability at: http://ws-i.org/profiles/basic/1.1/wsdl11.xsd

4.3 Types

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 2.2

The `wsdl:types` element of WSDL 1.1 encloses data type definitions that are relevant to the Web service described. The Profile places the following constraints pertinent to those portions of the content of the `wsdl:types` element that are referred to by WSDL elements that make Profile conformance claims:

4.3.1 QName References

XML Schema requires each QName reference to use either the target namespace, or an imported namespace (one marked explicitly with an `xsd:import` element). QName references to namespaces represented only by nested imports are not allowed.

WSDL 1.1 is unclear as to which schema target namespaces are suitable for QName references from a WSDL element. The Profile allows QName references from WSDL elements both to the target namespace defined by the `xsd:schema` element, and to imported namespaces. QName references to namespaces that are only defined through a nested import are not allowed.

R2101 *A* **DESCRIPTION** *MUST NOT use QName references to WSDL components in namespaces that have been neither imported, nor defined in the referring WSDL document.*

R2102 *A QName reference to a Schema component in a* **DESCRIPTION** *MUST use the namespace defined in the* `targetNamespace` *attribute on the* `xsd:schema` *element, or to a namespace defined in the* `namespace` *attribute on an* `xsd:import` *element within the* `xsd:schema` *element.*

4.3.2 Schema targetNamespace Structure

Requiring a targetNamespace on all xsd:schema elements that are children of `wsdl:types` is a good practice, places a minimal burden on authors of WSDL documents, and avoids the cases that are not as clearly defined as they might be.

R2105 *All* `xsd:schema` *elements contained in a* `wsdl:types` *element of a* **DESCRIPTION** *MUST have a* `targetNamespace` *attribute with a valid and non-null value, UNLESS the* `xsd:schema` *element has* `xsd:import` *and/or* `xsd:annotation` *as its only child element(s).*

4.3.3 soapenc:Array

The recommendations in WSDL 1.1 Section 2.2 for declaration of array types have been interpreted in various ways, leading to interoperability problems. Further, there are other clearer ways to declare arrays.

R2110 *In a* **DESCRIPTION**, *declarations MUST NOT extend or restrict the* `soapenc:Array` *type.*

R2111 *In a* **DESCRIPTION**, *declarations MUST NOT use* `wsdl:arrayType` *attribute in the type declaration.*

R2112 *In a* **DESCRIPTION**, *elements SHOULD NOT be named using the convention ArrayOfXXX.*

R2113 *An* **ENVELOPE** *MUST NOT include the* `soapenc:arrayType` *attribute.*

For example,

INCORRECT:

Given the WSDL Description:

```
<xsd:element name="MyArray2" type="tns:MyArray2Type"/>
<xsd:complexType name="MyArray2Type"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" >
  <xsd:complexContent>
     <xsd:restriction base="soapenc:Array">
       <xsd:sequence>
          <xsd:element name="x" type="xsd:string"
           minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute ref="soapenc:arrayType"
        wsdl:arrayType="tns:MyArray2Type[]"/>
   </xsd:restriction>
 </xsd:complexContent>
</xsd:complexType>
```

The envelope would serialize as (omitting namespace declarations for clarity):

```
<MyArray2 soapenc:arrayType="tns:MyArray2Type[]" >
  <x>abcd</x>
  <x>efgh</x>
</MyArray2>
```

CORRECT:

Given the WSDL Description:

```
<xsd:element name="MyArray1" type="tns:MyArray1Type"/>
<xsd:complexType name="MyArray1Type">
  <xsd:sequence>
   <xsd:element name="x" type="xsd:string"
    minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

The envelope would serialize as (omitting namespace declarations for clarity):

```
<MyArray1>
  <x>abcd</x>
  <x>efgh</x>
</MyArray1>
```

4.3.4 WSDL and Schema Definition Target Namespaces

The names defined by schemas and the names assigned to WSDL definitions are in separate symbol spaces.

R2114 *The target namespace for WSDL definitions and the target namespace for schema definitions in a* **DESCRIPTION** *MAY be the same.*WSDL20

4.4 Messages

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 2.3

In WSDL 1.1, `wsdl:message` elements are used to represent abstract definitions of the data being transmitted. It uses `wsdl:binding` elements to define how the abstract definitions are bound to a specific message serialization. The Profile places the following constraints on `wsdl:message` elements and on how conformant `wsdl:binding` elements may use `wsdl:message` element(s).

In this section the following definitions are used to make the requirements more compact and easier to understand.

Definition: rpc-literal binding

An "rpc-literal binding" is a wsdl:binding element whose child `wsdl:operation` elements are all rpc-literal operations.

An "rpc-literal operation" is a `wsdl:operation` child element of `wsdl:binding` whose `soapbind:body` descendant elements specify the use attribute with the value "literal", and either:

1. The style attribute with the value "rpc" is specified on the child `soapbind:operation` element; or
2. The `style` attribute is not present on the child `soapbind:operation` element, and the `soapbind:binding` element in the enclosing `wsdl:binding` specifies the style attribute with the value "rpc".

Definition: document-literal binding

A "document-literal binding" is a `wsdl:binding` element whose child `wsdl:operation` elements are all document-literal operations.

A "document-literal operation" is a `wsdl:operation` child element of `wsdl:binding` whose `soapbind:body` descendent elements specifies the `use` attribute with the value "literal" and, either:

1. The `style` attribute with the value "document" is specified on the child `soapbind:operation` element; or
2. The `style` attribute is not present on the child `soapbind:operation` element, and the `soapbind:binding` element in the enclosing `wsdl:binding` specifies the `style` attribute with the value "document"; or
3. The `style` attribute is not present on both the child `soapbind:operation` element and the `soapbind:binding` element in the enclosing `wsdl:binding`.

4.4.1 Bindings and Parts

There are various interpretations about how many `wsdl:part` elements are permitted or required for document-literal and rpc-literal bindings and how they must be defined.

R2201 *A document-literal binding in a* **DESCRIPTION** *MUST, in each of its `soapbind:body` element(s), have at most one part listed in the `parts` attribute, if the `parts` attribute is specified.*

R2209 *A `wsdl:binding` in a* **DESCRIPTION** *SHOULD bind every `wsdl:part` of a `wsdl:message` in the `wsdl:portType` to which it refers with a binding extension element.*

R2210 *If a document-literal binding in a* **DESCRIPTION** *does not specify the `parts` attribute on a `soapbind:body` element, the corresponding abstract `wsdl:message` MUST define zero or one `wsdl:part`s.*

R2202 *A `wsdl:binding` in a* **DESCRIPTION** *MAY contain `soapbind:body` element(s) that specify that zero parts form the `soap:Body`.*

R2203 *An rpc-literal binding in a* **DESCRIPTION** *MUST refer, in its `soapbind:body` element(s), only to `wsdl:part` element(s) that have been defined using the `type` attribute.*

R2211 *An* **ENVELOPE** *described with an rpc-literal binding MUST NOT have the `xsi:nil` attribute with a value of "1" or "true" on the part accessors.*

R2207 *A `wsdl:message` in a* **DESCRIPTION** *MAY contain `wsdl:part`s that use the `elements` attribute provided those* *`wsdl:part`s are not referred to by a `soapbind:body` in an rpc-literal binding.*

R2204 *A document-literal binding in a* **DESCRIPTION** *MUST refer, in each of its `soapbind:body` element(s), only to `wsdl:part` element(s) that have been defined using the `element` attribute.*

R2208 *A binding in a* **DESCRIPTION** *MAY contain `soapbind:header` element(s) that refer to `wsdl:part`s in the same `wsdl:message` that are referred to by its `soapbind:body` element(s).*

R2212 *An* **ENVELOPE** *MUST contain exactly one part accessor element for each of the `wsdl:part` elements bound to the envelope's corresponding `soapbind:body` element.*

R2213 *In a doc-literal description where the value of the parts attribute of soapbind:body is an empty string, the corresponding* **ENVELOPE** *MUST have no element content in the soap:Body element.*

R2214 *In a rpc-literal description where the value of the parts attribute of `soapbind:body` is an empty string, the corresponding* **ENVELOPE** *MUST have no part accessor elements.*

Use of `wsdl:message` elements with zero parts is permitted in Document styles to permit operations that can send or receive envelopes with empty `soap:Body`s. Use of `wsdl:message` elements with zero parts is permitted in RPC styles to permit operations that have no (zero) parameters and/or a return value.

For document-literal bindings, the Profile requires that at most one part, abstractly defined with the `element` attribute, be serialized into the `soap:Body` element.

When a `wsdl:part` element is defined using the `type` attribute, the serialization of that part in a message is equivalent to an implicit (XML Schema) qualification of a `minOccurs` attribute with the value "1", a `maxOccurs` attribute with the value "1" and a `nillable` attribute with the value "false".

It is necessary to specify the equivalent implicit qualification because the `wsdl:part` element does not allow one to specify the cardinality and nillability rules. Specifying the cardinality and the nillability rules facilitates interoperability between implementations. The equivalent implicit qualification for nillable attribute has a value of "false" because if it is specified to be "true" one cannot design a part whereby the client is always required to send a value. For applications that want to allow the `wsdl:part` to to be nillable, it is expected that applications will generate a complexType wrapper and specify the nillability rules for the contained elements of such a wrapper.

4.4.2 Bindings and Faults

There are several interpretations for how `wsdl:part` elements that describe `soapbind:fault`, `soapbind:header`, and `soapbind:headerfault` may be defined.

R2205 *A `wsdl:binding` in a **DESCRIPTION** MUST refer, in each of its `soapbind:header`, `soapbind:headerfault` and `soapbind:fault` elements, only to `wsdl:part` element(s) that have been defined using the `element` attribute.*

Because faults and headers do not contain parameters, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` assume, per WSDL 1.1, that the value of the `style` attribute is "document". R2204 requires that all `wsdl:part` elements with a `style` attribute whose value is "document" that are bound to `soapbind:body` be defined using the `element` attribute. This requirement does the same for `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements.

4.4.3 Declaration of part Elements

Examples 4 and 5 in WSDL 1.1 Section 3.1 incorrectly show the use of XML Schema types (e.g. "xsd:string") as a valid value for the `element` attribute of a `wsdl:part` element.

R2206 *A `wsdl:message` in a **DESCRIPTION** containing a `wsdl:part` that uses the `element` attribute MUST refer, in that attribute, to a global element declaration.*

For example,

INCORRECT:
```
<message name="GetTradePriceInput">
    <part name="tickerSymbol" element="xsd:string"/>
    <part name="time" element="xsd:timeInstant"/>
</message>
```

INCORRECT:
```
<message name="GetTradePriceInput">
    <part name="tickerSymbol" element="xsd:string"/>
</message>
```

CORRECT:
```
<message name="GetTradePriceInput">
    <part name="body" element="tns:SubscribeToQuotes"/>
</message>
```

4.5 Port Types

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 2.4

In WSDL 1.1, `wsdl:portType` elements are used to group a set of abstract operations. The Profile places the following constraints on conformant `wsdl:portType` element(s):

4.5.1 Ordering of part Elements

Permitting the use of `parameterOrder` helps code generators in mapping between method signatures and messages on the wire.

R2301 *The order of the elements in the `soap:Body` of an* **ENVELOPE** *MUST be the same as that of the `wsdl:parts` in the `wsdl:message` that describes it for each of the `wsdl:part` elements bound to the envelope's corresponding `soapbind:body` element.*

R2302 *A* **DESCRIPTION** *MAY use the `parameterOrder` attribute of an `wsdl:operation` element to indicate the return value and method signatures as a hint to code generators.*

4.5.2 Allowed Operations

Solicit-Response and Notification operations are not well defined by WSDL 1.1; furthermore, WSDL 1.1 does not define bindings for them.

R2303 *A* **DESCRIPTION** *MUST NOT use Solicit-Response and Notification type operations in a `wsdl:portType` definition.*

4.5.3 Distinctive Operations

Operation name overloading in a `wsdl:portType` is disallowed by the Profile.

R2304 *A `wsdl:portType` in a* **DESCRIPTION** *MUST have operations with distinct values for their `name` attributes.*

Note that this requirement applies only to the `wsdl:operation`s within a given `wsdl:portType`. A `wsdl:portType` may have `wsdl:operation`s with names that are the same as those found in other `wsdl:portType`s.

4.5.4 parameterOrder Attribute Construction

WSDL 1.1 does not clearly state how the `parameterOrder` attribute of the `wsdl:operation` element (which is the child of the `wsdl:portType` element) should be constructed.

R2305 *A `wsdl:operation` element child of a `wsdl:portType` element in a* **DESCRIPTION** *MUST be constructed so that the `parameterOrder` attribute, if present, omits at most 1 `wsdl:part` from the output message.*

If a `wsdl:part` from the output message is omitted from the list of `wsdl:part`s that is the value of the `parameterOrder` attribute, the single omitted `wsdl:part` is the return value. There are no restrictions on the type of the return value. If no part is omitted, there is no return value.

4.5.5 Exclusivity of type and element Attributes

WSDL 1.1 does not clearly state that both `type` and `element` attributes cannot be specified to define a `wsdl:part` in a `wsdl:message`.

R2306 *A `wsdl:message` in a* **DESCRIPTION** *MUST NOT specify both `type` and `element` attributes on the same `wsdl:part`.*

4.6 Bindings

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 2.5

In WSDL 1.1, the `wsdl:binding` element supplies the concrete protocol and data format specifications for the operations and messages defined by a particular `wsdl:portType`. The Profile places the following constraints on conformant binding specifications:

4.6.1 Use of SOAP Binding

The Profile limits the choice of bindings to the well-defined and most commonly used SOAP binding.

R2401 *A `wsdl:binding` element in a* **DESCRIPTION** *MUST use WSDL SOAP Binding as defined in WSDL 1.1 Section 3.*

Note that this places a requirement on the construction of conformant `wsdl:binding` elements. It does not place a requirement on descriptions as a whole; in particular, it does not preclude WSDL documents from containing non-conformant `wsdl:binding` elements. Also, a binding may have WSDL extensibility elements present which change how messages are serialized.

4.7 SOAP Binding

The following specifications (or sections thereof) are referred to in this section of the Profile:

- WSDL 1.1, Section 3.0

WSDL 1.1 defines a binding for SOAP 1.1 endpoints. The Profile mandates the use of SOAP binding as defined in WSDL 1.1, and places the following constraints on its use:

4.7.1 Specifying the transport Attribute

There is an inconsistency between the WSDL 1.1 specification and the WSDL 1.1 schema regarding the `transport` attribute. The WSDL 1.1 specification requires it; however, the schema shows it to be optional.

R2701 *The `wsdl:binding` element in a* **DESCRIPTION** *MUST be constructed so that its `soapbind:binding` child element specifies the `transport` attribute.*

4.7.2 HTTP Transport

The profile limits the underlying transport protocol to HTTP.

R2702 *A `wsdl:binding` element in a* **DESCRIPTION** *MUST specify the HTTP transport protocol with SOAP binding.*

Specifically, the `transport` attribute of its `soapbind:binding` child MUST have the value "http://schemas.xmlsoap.org/soap/http".

Note that this requirement does not prohibit the use of HTTPS; See R5000.

4.7.3 Consistency of style Attribute

The `style`, "document" or "rpc", of an interaction is specified at the `wsdl:operation` level, permitting `wsdl:binding`s whose `wsdl:operation`s have different `style`s. This has led to interoperability problems.

R2705 *A `wsdl:binding` in a* **DESCRIPTION** *MUST either be a rpc-literal binding or a document-literal binding.*

4.7.4 Encodings and the use Attribute

The Profile prohibits the use of encodings, including the SOAP encoding.

R2706 *A `wsdl:binding` in a* **DESCRIPTION** *MUST use the value of "literal" for the `use` attribute in all `soapbind:body`, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements.*

4.7.5 Multiple Bindings for portType Elements

The Profile explicitly permits multiple bindings for the same portType.

R2709 *A `wsdl:portType` in a* **DESCRIPTION** *MAY have zero or more `wsdl:binding`s that refer to it, defined in the same or other WSDL documents.*

4.7.6 Operation Signatures

Definition: operation signature

The profile defines the "operation signature" to be the fully qualified name of the child element of SOAP body of the SOAP input message described by an operation in a WSDL binding.

In the case of rpc-literal binding, the operation name is used as a wrapper for the part accessors. In the document-literal case, since a wrapper with the operation name is not present, the message signatures must be correctly designed so that they meet this requirement.

An endpoint that supports multiple operations must unambiguously identify the operation being invoked based on the input message that it receives. This is only possible if all the operations specified in the `wsdl:binding` associated with an endpoint have a unique operation signature.

R2710 *The operations in a `wsdl:binding` in a* **DESCRIPTION** *MUST result in operation signatures that are different from one another.*

4.7.7 Multiple Ports on an Endpoint

When input messages destined for two different `wsdl:port`s at the same network endpoint are indistinguishable on the wire, it may not be possible to determine the `wsdl:port` being invoked by them. This may cause interoperability problems. However, there may be situations (e.g., SOAP versioning, application versioning, conformance to different profiles) where it is desirable to locate more than one port on an endpoint; therefore, the Profile allows this.

R2711 *A* **DESCRIPTION** *SHOULD NOT have more than one `wsdl:port` with the same value for the `location` attribute of the `soapbind:address` element.*

4.7.8 Child Element for Document-Literal Bindings

WSDL 1.1 is not completely clear what, in document-literal style bindings, the child element of `soap:Body` is.

R2712 *A document-literal binding MUST be serialized as an* **ENVELOPE** *with a `soap:Body` whose child element is an instance of the global element declaration referenced by the corresponding `wsdl:message` part.*

4.7.9 One-Way Operations

There are differing interpretations of how HTTP is to be used when performing one-way operations.

R2714 *For one-way operations, an* **INSTANCE** *MUST NOT return a HTTP response that contains an envelope. Specifically, the HTTP response entity-body must be empty.*

R2750 *A* **CONSUMER** *MUST ignore an envelope carried in a HTTP response message in a one-way operation.*

R2727 *For one-way operations, a* **CONSUMER** *MUST NOT interpret a successful HTTP response status code (i.e., 2xx) to mean the message is valid or that the receiver would process it.*

One-way operations do not produce SOAP responses. Therefore, the Profile prohibits sending a SOAP envelope in response to a one-way operation. This means that transmission of one-way operations can not result in processing level responses or errors. For example, a "500 Internal Server Error" HTTP response that contains a fault can not be returned in this situation.

The HTTP response to a one-way operation indicates the success or failure of the transmission of the message. Based on the semantics of the different response status codes supported by the HTTP protocol, the Profile specifies that "200" and "202" are the preferred status codes that the sender should expect, signifying that the one-way message was received. A successful transmission does not indicate that the SOAP processing layer and the application logic has had a chance to validate the envelope or have committed to processing it.

4.7.10 Namespaces for soapbind Elements

There is confusion about what namespace is associated with the child elements of various children of `soap:Envelope`, which has led to interoperability difficulties. The Profile defines these.

R2716 *A document-literal binding in a* **DESCRIPTION** *MUST NOT have the `namespace` attribute specified on contained `soapbind:body`, `soapbind:header`, `soapbind:headerfault` and `soapbind:fault` elements.*

R2717 *An rpc-literal binding in a* **DESCRIPTION** *MUST have the `namespace` attribute specified, the value of which MUST be an absolute URI, on contained `soapbind:body` elements.*

R2726 *An rpc-literal binding in a* **DESCRIPTION** *MUST NOT have the `namespace` attribute specified on contained `soapbind:header`, `soapbind:headerfault` and `soapbind:fault` elements.*

In a document-literal SOAP binding, the serialized element child of the `soap:Body` gets its namespace from the targetNamespace of the schema that defines the element. Use of the `namespace` attribute of the `soapbind:body` element would override the element's namespace. This is not allowed by the Profile.

Conversely, in a rpc-literal SOAP binding, the serialized child element of the `soap:Body` element consists of a wrapper element, whose namespace is the value of the `namespace` attribute of the `soapbind:body` element and whose local name is either the name of the operation or the name of the operation suffixed with "Response". The `namespace` attribute is required, as opposed to being optional, to ensure that the children of the `soap:Body` element are namespace-qualified.

4.7.11 Consistency of portType and binding Elements

The WSDL description must be consistent at both `wsdl:portType` and `wsdl:binding` levels.

R2718 *A `wsdl:binding` in a* **DESCRIPTION** *MUST have the same set of `wsdl:operation`s as the `wsdl:portType` to which it refers.* C

4.7.12 Describing headerfault Elements

There is inconsistency between WSDL specification text and the WSDL schema regarding `soapbind:headerfault`s.

R2719 *A `wsdl:binding` in a* **DESCRIPTION** *MAY contain no `soapbind:headerfault` elements if there are no known header faults.*

The WSDL 1.1 schema makes the specification of `soapbind:headerfault` element mandatory on `wsdl:input` and `wsdl:output` elements of an operation, whereas the WSDL 1.1 specification marks them optional. The specification is correct.

4.7.13 Enumeration of Faults

A Web service description should include all faults known at the time the service is defined. There is also need to permit generation of new faults that had not been identified when the Web service was defined.

R2740 *A `wsdl:binding` in a* **DESCRIPTION** *SHOULD contain a `soapbind:fault` describing each known fault.*

R2741 *A `wsdl:binding` in a* **DESCRIPTION** *SHOULD contain a `soapbind:headerfault` describing each known header fault.*

R2742 *An* **ENVELOPE** *MAY contain fault with a `detail` element that is not described by a `soapbind:fault` element in the corresponding WSDL description.*

R2743 *An* **ENVELOPE** *MAY contain the details of a header processing related fault in a SOAP header block that is not described by a `soapbind:headerfault` element in the corresponding WSDL description.*

4.7.14 Type and Name of SOAP Binding Elements

The WSDL 1.1 schema disagrees with the WSDL 1.1 specification about the name and type of an attribute of the `soapbind:header` and `soapbind:headerfault` elements.

R2720 *A `wsdl:binding` in a* **DESCRIPTION** *MUST use the `part` attribute with a schema type of "NMTOKEN" on all contained `soapbind:header` and `soapbind:headerfault` elements.*

R2749 *A `wsdl:binding` in a* **DESCRIPTION** *MUST NOT use the `parts` attribute on contained `soapbind:header` and `soapbind:headerfault` elements.*

The WSDL Schema gives the attribute's name as "parts" and its type as "NMTOKENS". The schema is incorrect since each `soapbind:header` and `soapbind:headerfault` element references a single `wsdl:part`.

For example,

CORRECT:
```
<binding name="StockQuoteSoap" type="tns:StockQuotePortType">
  <soapbind:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="SubscribeToQuotes">
      <input message="tns:SubscribeToQuotes">
        <soapbind:body parts="body" use="literal"/>
        <soapbind:header message="tns:SubscribeToQuotes"
                        part="subscribeheader" use="literal"/>
     </input>
   </operation>
</binding>
```

4.7.15 name Attribute on Faults

There is inconsistency between the WSDL 1.1 specification and the WSDL 1.1 schema, which does not list the `name` attribute.

R2721 *A `wsdl:binding` in a* **DESCRIPTION** *MUST have the `name` attribute specified on all contained `soapbind:fault` elements.*

R2754 *In a* **DESCRIPTION***, the value of the `name` attribute on a `soapbind:fault` element MUST match the value of the `name` attribute on its parent `wsdl:fault` element.*

4.7.16 Omission of the use Attribute

There is inconsistency between the WSDL 1.1 specification and the WSDL 1.1 schema regarding the `use` attribute.

R2722 *A `wsdl:binding` in a* **DESCRIPTION** *MAY specify the `use` attribute on contained `soapbind:fault` elements.* C

R2723 *If in a `wsdl:binding` in a* **DESCRIPTION** *the `use` attribute on a contained `soapbind:fault` element is present, its value MUST be "literal".*

WSDL 1.1 Section 3.6 indicates that the `use` attribute of `soapbind:fault` is required while in the schema the `use` attribute is defined as optional. The Profile defines it as optional, to be consistent with `soapbind:body`.

Since the `use` attribute is optional, the Profile identifies the default value for the attribute when omitted.

Finally, to assure that the Profile is self-consistent, the only permitted value for the `use` attribute is "literal".

4.7.17 Default for use Attribute

There is an inconsistency between the WSDL 1.1 specification and the WSDL 1.1 schema regarding whether the `use` attribute is optional on `soapbind:body`, `soapbind:header`, and `soapbind:headerfault`, and if so, what omitting the attribute means.

R2707 *A `wsdl:binding` in a* **DESCRIPTION** *that contains one or more `soapbind:body`, `soapbind:fault`, `soapbind:header` or `soapbind:headerfault` elements that do not specify the `use` attribute MUST be interpreted as though the value "literal" had been specified in each case.*

4.7.18 Consistency of Envelopes with Descriptions

These requirements specify that when an instance receives an envelope that does not conform to the WSDL description, a fault should be generated unless the instance takes it upon itself to process the envelope regardless of this.

As specified by the SOAP processing model, (a) a "VersionMismatch" faultcode must be generated if the namespace of the "Envelope" element is incorrect, (b) a "MustUnderstand" fault must be generated if the instance does not understand a

SOAP header block with a value of "1" for the `soap:mustUnderstand` attribute. In all other cases where an envelope is inconsistent with its WSDL description, a fault with a "Client" faultcode should be generated.

R2724 *If an* **INSTANCE** *receives an envelope that is inconsistent with its WSDL description, it SHOULD generate a* `soap:Fault` *with a faultcode of "Client", unless a "MustUnderstand" or "VersionMismatch" fault is generated.*

R2725 *If an* **INSTANCE** *receives an envelope that is inconsistent with its WSDL description, it MUST check for "VersionMismatch", "MustUnderstand" and "Client" fault conditions in that order.*

4.7.19 Response Wrappers

WSDL 1.1 Section 3.5 could be interpreted to mean the RPC response wrapper element must be named identical to the name of the `wsdl:operation`.

R2729 *An* **ENVELOPE** *described with an rpc-literal binding that is a response MUST have a wrapper element whose name is the corresponding* `wsdl:operation` *name suffixed with the string "Response".*

4.7.20 Part Accessors

For rpc-literal envelopes, WSDL 1.1 is not clear what namespace, if any, the accessor elements for parameters and return value are a part of. Different implementations make different choices, leading to interoperability problems.

R2735 *An* **ENVELOPE** *described with an rpc-literal binding MUST place the part accessor elements for parameters and return value in no namespace.*

R2755 *The part accessor elements in a* **MESSAGE** *described with an rpc-literal binding MUST have a local name of the same value as the* `name` *attribute of the corresponding* `wsdl:part element.`

Settling on one alternative is crucial to achieving interoperability. The Profile places the part accessor elements in no namespace as doing so is simple, covers all cases, and does not lead to logical inconsistency.

4.7.21 Namespaces for Children of Part Accessors

For rpc-literal envelopes, WSDL 1.1 is not clear on what the correct namespace qualification is for the child elements of the part accessor elements when the corresponding abstract parts are defined to be of types from a different namespace than the `targetNamespace` of the WSDL description for the abstract parts.

R2737 *An* **ENVELOPE** *described with an rpc-literal binding MUST namespace qualify the descendents of part accessor elements for the parameters and the return value, as* *defined by the schema in which the part accessor types are defined.*

WSDL 1.1 Section 3.5 states: "The part names, types and value of the namespace attribute are all inputs to the encoding, although the namespace attribute only applies to content not explicitly defined by the abstract types."

However, it does not explicitly state that the element and attribute content of the abstract (complexType) types is namespace qualified to the targetNamespace in which those elements and attributes were defined. WSDL 1.1 was intended to function in much the same manner as XML Schema. Hence, implementations must follow the same rules as for XML Schema. If a complexType defined in `targetNamespace` "A" were imported and referenced in an element declaration in a schema with `targetNamespace` "B", the element and attribute content of the child elements of that complexType would be qualified to namespace "A" and the element would be qualified to namespace "B".

For example,

CORRECT:

Given this WSDL, which defines some schema in the "http://example.org/foo/" namespace in the `wsdl:types` section contained within a `wsdl:definitions` that has a `targetNamespace` attribute with the value "http://example.org/bar/" (thus, having a type declared in one namespace and the containing element defined in another);

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bar="http://example.org/bar/"
targetNamespace="http://example.org/bar/"
xmlns:foo="http://example.org/foo/">
<types>
   <xsd:schema targetNamespace="http://example.org/foo/"
       xmlns:tns="http://example.org/foo/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified"
       attributeFormDefault="unqualified">
       <xsd:complexType name="fooType">
          <xsd:sequence>
             <xsd:element ref="tns:bar"/>
             <xsd:element ref="tns:baf"/>
          </xsd:sequence>
       </xsd:complexType>
       <xsd:element name="bar" type="xsd:string"/>
       <xsd:element name="baf" type="xsd:integer"/>
   </xsd:schema>
</types>
<message name="BarMsg">
   <part name="BarAccessor" type="foo:fooType"/>
</message>
<portType name="BarPortType">
   <operation name="BarOperation">
     <input message="bar:BarMsg"/>
   </operation>
</portType>
<binding name="BarSOAPBinding" type="bar:BarPortType">
   <soapbind:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="rpc"/>
   <operation name="BarOperation">
     <input>
       <soapbind:body use="literal" namespace="http://example.org/bar/"/>
     </input>
   </operation>
</binding>
<service name="serviceName">
  <port name="BarSOAPPort" binding="bar:BarSOAPBinding">
    <soapbind:address location="http://example.org/myBarSOAPPort"/>
  </port>
</service>
</definitions>
```

The resulting envelope for BarOperation is:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:foo="http://example.org/foo/">
  <s:Header/>
    <s:Body>
      <m:BarOperation xmlns:m="http://example.org/bar/">
         <BarAccessor>
            <foo:bar>String</foo:bar>
            <foo:baf>0</foo:baf>
         </BarAccessor>
      </m:BarOperation>
    </s:Body>
</s:Envelope>
```

4.7.22 Required Headers

WSDL 1.1 does not clearly specify whether all `soapbind:header`s specified on the `wsdl:input` or `wsdl:output` elements of a `wsdl:operation` element in the SOAP binding section of a WSDL description must be included in the resultant envelopes when they are transmitted. The Profile makes all such headers mandatory, as there is no way in WSDL 1.1 to mark a header optional.

R2738 *An* **ENVELOPE** *MUST include all `soapbind:header`s specified on a `wsdl:input` or `wsdl:output` of a `wsdl:operation` of a `wsdl:binding` that describes it.*

4.7.23 Allowing Undescribed Headers

Headers are SOAP's extensibility mechanism. Headers that are not defined in the WSDL description may need to be included in the envelopes for various reasons.

R2739 *An* **ENVELOPE** *MAY contain SOAP header blocks that are not described in the `wsdl:binding` that describes it.*

R2753 *An* **ENVELOPE** *containing SOAP header blocks that are not described in the appropriate `wsdl:binding` MAY have the `mustUnderstand` attribute on such SOAP header blocks set to '1'.*

4.7.24 Ordering Headers

There is no correlation between the order of `soapbind:header`s in the description and the order of SOAP header blocks in the envelope. Similarly, more than one instance of each specified SOAP header block may occur in the envelope.

R2751 *The order of `soapbind:header` elements in `soapbind:binding` sections of a* **DESCRIPTION** *MUST be considered independent of the order of SOAP header blocks in the envelope.*

R2752 *An* **ENVELOPE** *MAY contain more than one instance of each SOAP header block for each `soapbind:header` element in the appropriate child of `soapbind:binding` in the corresponding description.*

4.7.25 Describing SOAPAction

Interoperability testing has demonstrated that requiring the `SOAPAction` HTTP header field-value to be quoted increases interoperability of implementations. Even though HTTP allows for header field-values to be unquoted, some implementations require that the value be quoted.

The `SOAPAction` header is purely a hint to processors. All vital information regarding the intent of a message is carried in the envelope.

R2744 *A HTTP request* **MESSAGE** *MUST contain a `SOAPAction` HTTP header field with a quoted value equal to the value of the `soapAction` attribute of `soapbind:operation`, if present in the corresponding WSDL description.*

R2745 *A HTTP request* **MESSAGE** *MUST contain a `SOAPAction` HTTP header field with a quoted empty string value, if in the corresponding WSDL description, the `soapAction` of `soapbind:operation` is either not present, or present with an empty string as its value.*

See also R1119 and related requirements for more discussion of SOAPAction.

For example,

CORRECT:

A WSDL Description that has:

```
<soapbind:operation soapAction="foo" />
```

results in a message with a corresponding SOAPAction HTTP header field as follows:

```
SOAPAction: "foo"
```

CORRECT:

A WSDL Description that has:

```
<soapbind:operation />
```

or

```
<soapbind:operation soapAction="" />
```

results in a message with a corresponding SOAPAction HTTP header field as follows:

```
SOAPAction: ""
```

4.7.26 SOAP Binding Extensions

The `wsdl:required` attribute has been widely misunderstood and used by WSDL authors sometimes to incorrectly indicate the optionality of `soapbind:header`s. The `wsdl:required` attribute, as specified in WSDL1.1, is an extensibility mechanism aimed at WSDL processors. It allows new WSDL extension elements to be introduced in a graceful manner. The intent of `wsdl:required` is to signal to the WSDL processor whether the extension element needs to be recognized and understood by the WSDL processor in order that the WSDL description be correctly processed. It is not meant to signal conditionality or optionality of some construct that is included in the envelopes. For example, a `wsdl:required` attribute with the value "false" on a `soapbind:header` element must not be interpreted to signal to the WSDL processor that the described SOAP header block is conditional or optional in the envelopes generated from the WSDL description. It is meant to be interpreted as "in order to send a envelope to the endpoint that includes in its description the `soapbind:header` element, the WSDL processor MUST understand the semantic implied by the `soapbind:header` element."

The default value for the `wsdl:required` attribute for WSDL 1.1 SOAP Binding extension elements is "false". Most WSDL descriptions in practice do not specify the `wsdl:required` attribute on the SOAP Binding extension elements, which could be interpreted by WSDL processors to mean that the extension elements may be ignored. The Profile requires that all WSDL SOAP 1.1 extensions be understood and processed by the consumer, irrespective of the presence or the value of the `wsdl:required` attribute on an extension element.

R2747 *A* **CONSUMER** *MUST understand and process all WSDL 1.1 SOAP Binding extension elements, irrespective of the presence or absence of the* `wsdl:required` *attribute on an extension element; and irrespective of the value of the* `wsdl:required` *attribute, when present.*

R2748 *A* **CONSUMER** *MUST NOT interpret the presence of the* `wsdl:required` *attribute on a* `soapbind` *extension element with a value of "false" to mean the extension element is optional in the envelopes generated from the WSDL description.*

4.8 Use of XML Schema

The following specifications (or sections thereof) are referred to in this section of the Profile:

- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes

WSDL 1.1 uses XML Schema as one of its type systems. The Profile mandates the use of XML Schema as the type system for WSDL descriptions of Web Services.

R2800 *A* **DESCRIPTION** *MAY use any construct from XML Schema 1.0.*

R2801 *A* **DESCRIPTION** *MUST use XML Schema 1.0 Recommendation as the basis of user defined datatypes and structures.*

5 Service Publication and Discovery

When publication or discovery of Web services is required, UDDI is the mechanism the Profile has adopted to describe Web service providers and the Web services they provide. Business, intended use, and Web service type descriptions are made in UDDI terms; detailed technical descriptions are made in WSDL terms. Where the two specifications define overlapping descriptive data and both forms of description are used, the Profile specifies that the descriptions must not conflict.

Registration of Web service instances in UDDI registries is optional. By no means do all usage scenarios require the kind of metadata and discovery UDDI provides, but where such capability is needed, UDDI is the sanctioned mechanism.

Note that the Web services that constitute UDDI V2 are not fully conformant with the Profile 1.0 because they do not accept messages whose envelopes are encoded in either UTF-8 and UTF-16 as required by the Profile. (They accept UTF-8 only.) That there should be such a discrepancy is hardly surprising given that UDDI V2 was designed and, in many cases, implemented before the Profile was developed. UDDI's designers are aware of UDDI V2's nonconformance and will take it into consideration in their future work.

This section of the Profile incorporates the following specifications by reference:

- UDDI Version 2.04 API Specification, Dated 19 July 2002
- UDDI Version 2.03 Data Structure Reference, Dated 19 July 2002
- UDDI Version 2 XML Schema

5.1 bindingTemplates

The following specifications (or sections thereof) are referred to in this section of the Profile:

- UDDI Version 2.03 Data Structure Reference, Section 7