Basic Profile Version 1.1

Working Group Draft

Date: 2003/12/06 07:48:25

This revision:
http://www.ws-i.org/Profiles/Basic/2003-12/BasicProfile-1.1.htm
Editors:
Keith Ballinger, Microsoft (1.0)
David Ehnebuske, IBM (1.0)
Martin Gudgin, Microsoft (1.0)
Mark Nottingham, BEA Systems
Prasad Yendluri, webMethods (1.0)
Administrative contact:
secretary@ws-i.org

Abstract

This document defines the WS-I Basic Profile 1.1, consisting of a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

Status of this Document

This document is a Working Group Draft; it has been accepted by the Working Group as reflecting the current state of discussions. It is a work in progress, and should not be considered authoritative or final; other documents may supersede this document.

Feedback

The Web Services-Interoperability Organization (WS-I) would like to receive input, suggestions and other feedback ("Feedback") on this work from a wide variety of industry participants to improve its quality over time.

By sending email, or otherwise communicating with WS-I, you (on behalf of yourself if you are an individual, and your company if you are providing Feedback on behalf of the company) will be deemed to have granted to WS-I, the members of WS-I, and other parties that have access to your Feedback, a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free license to use, disclose, copy, license, modify, sublicense or otherwise distribute and exploit in any manner whatsoever the Feedback you provide regarding the work. You acknowledge that you have no expectation of confidentiality with respect to any Feedback you provide. You represent and warrant that you have rights to provide this Feedback, and if you are providing Feedback on behalf of a company, you represent and warrant that you have the rights to provide Feedback on behalf of your company. You also acknowledge that WS-I is not required to review, discuss, use, consider or in any way incorporate your Feedback into future versions of its work. If WS-I does incorporate some or all of your Feedback in a future version of the work, it may, but is not obligated to include your name (or, if you are identified as acting on behalf of your company, the name of your company) on a list of contributors to the work. If the foregoing is not acceptable to you and any company on whose behalf you are acting, please do not provide any Feedback.

Feedback on this document should be directed to wsbasic_comment@ws-i.org.

Table of Contents

1. Introduction
1.1. Relationships to Other Profiles
1.2. Notational Conventions
2. Profile Conformance
2.1. Conformance Requirements
2.2. Conformance Targets
2.3. Conformance Scope
2.4. Claiming Conformance
3. Messaging
3.1. SOAP Envelopes
3.1.1. SOAP Envelope Structure
3.1.2. SOAP Envelope Namespace

3.1.3. SOAP Body Namespace Qualification
3.1.4. Disallowed Constructs
3.1.5. SOAP Trailers
3.1.6. SOAP encodingStyle Attribute
3.1.7. SOAP mustUnderstand Attribute
3.1.8. xsi:type Attributes
3.2. SOAP Processing Model
3.2.1. Mandatory Headers
3.2.2. Generating mustUnderstand Faults
3.2.3. SOAP Fault Processing
3.3. SOAP Faults
3.3.1. SOAP Fault Structure
3.3.2. SOAP Fault Namespace Qualification
3.3.3. SOAP Fault Extensibility
3.3.4. SOAP Fault Language
3.3.5. SOAP Custom Fault Codes
3.3.6. Identifying SOAP Faults
3.4. Use of SOAP in HTTP
3.4.1. HTTP Protocol Binding
3.4.2. HTTP Methods and Extensions
3.4.3. SOAPAction Header Syntax
3.4.4. HTTP and TCP Ports
3.4.5. HTTP Success Status Codes
3.4.6. HTTP Redirect Status Codes
3.4.7. HTTP Client Error Status Codes
3.4.8. HTTP Server Error Status Codes
3.4.9. HTTP Cookies
4. Service Description
4.1. Required Description
4.2. Document Structure
4.2.1. WSDL Schema Definitions
4.2.2. WSDL and Schema Import
4.2.3. WSDL Import location Attribute Structure
4.2.4. WSDL Import location Attribute Semantics
4.2.5. Placement of WSDL import Elements
4.2.6. XML Version Requirements
4.2.7. WSDL and the Unicode BOM
4.2.8. Acceptable WSDL Character Encodings
4.2.9. Namespace Coercion
4.2.10. WSDL documentation Element
4.2.11. WSDL Extensions
4.3. Types
4.3.1. QName References
4.3.2. Schema targetNamespace Structure
4.3.3. soapenc:Array
4.3.4. WSDL and Schema Definition Target Namespaces

4.4. Messages
4.4.1. Bindings and Parts
4.4.2. Bindings and Faults
4.4.3. Unbound portType Element Contents
4.4.4. Declaration of part Elements
4.5. Port Types
4.5.1. Ordering of part Elements
4.5.2. Allowed Operations
4.5.3. Distinctive Operations
4.5.4. parameterOrder Attribute Construction
4.5.5. Exclusivity of type and element Attributes
4.6. Bindings
4.6.1. Use of SOAP Binding
4.7. SOAP Binding
4.7.1. Specifying the transport Attribute
4.7.2. HTTP Transport
4.7.3. Consistency of style Attribute
4.7.4. Encodings and the use Attribute
4.7.5. Default for use Attribute
4.7.6. Multiple Bindings for portType Elements
4.7.7. Wire Signatures for Operations
4.7.8. Multiple Ports on an Endpoint
4.7.9. Child Element for Document-Literal Bindings
4.7.10. One-Way Operations
4.7.11. Namespaces for soapbind Elements
4.7.12. Consistency of portType and binding Elements
4.7.13. Describing headerfault Elements
4.7.14. Enumeration of Faults
4.7.15. Type and Name of SOAP Binding Elements
4.7.16. name Attribute on Faults
4.7.17. Omission of the use Attribute
4.7.18. Consistency of Envelopes with Descriptions
4.7.19. Response Wrappers
4.7.20. Namespace for Part Accessors
4.7.21. Namespaces for Children of Part Accessors
4.7.22. Required Headers
4.7.23. Allowing Undescribed Headers
4.7.24. Ordering Headers
4.7.25. Describing SOAPAction
4.7.26. SOAP Binding Extensions
4.8. Use of XML Schema
5. Service Publication and Discovery
5.1. bindingTemplates
5.2. tModels
6. Security
6.1. Use of HTTPS

Appendix I: Referenced Specifications
Appendix II: Extensibility Points
Appendix III: Acknowledgements

1. Introduction

This document defines the WS-I Basic Profile 1.1 (hereafter, "Profile"), consisting of a set of non-proprietary Web services specifications, along with clarifications to 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 document and those in the referenced specifications.

1.1 Relationships to Other Profiles

This Profile is derived from the Basic Profile 1.0, 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, so that a profile for attachments could be composed with the Basic Profile 1.1.

This Profile is intended to supersede Basic Profile 1.0.

The manner in which this profile supersedes BP 1.0 is currently under discussion.

A combined claim of conformance to both the Basic Profile 1.1 and the Simple SOAP Binding Profile 1.0 should be equivalent to a claim of conformance to 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.

1.2 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

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

> Rnnnn*Statement text here.*

where "nnnn" is replaced by the statement number. Each statement contains exactly one requirement level keyword (e.g., "MUST") and one conformance target keyword (e.g., "MESSAGE").

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

Some statements 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., "SOAP12" for SOAP Version 1.2). Note that because such work is not complete, the specification that the requirement is derived from may change; this information is included only as a convenience to implementers.

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"

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 the Profile defines conformance. See the Profile Conformance Framework for more information about conformance to WS-I profiles.

2.1 Conformance Requirements

Requirements state the criteria for conformance to the Profile. They typically refer to an existing specification and embody refinements, 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 has exactly one conformance target and one requirement level, to avoid ambiguity. Additional text may be included to illuminate requirements or group of requirements (e.g., rationale and examples); however, requirement statements alone should be considered in determining conformance.

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.

This Profile defines the following conformance targets:

- **ENVELOPE** - the serialization of the `soap:Envelope` element and its content.
- **MESSAGE** - protocol elements that transport the ENVELOPE (e.g., SOAP/HTTP messages).
- **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).
- **REGDATA** - registry elements that are involved in the registration and discovery of Web services (e.g. UDDI tModels).
- **INSTANCE** - software that implements a `wsdl:port` or a `uddi:bindingTemplate`.
- **CONSUMER** - software that invokes an INSTANCE
- **SENDER** - software that generates a message according to the protocol(s) associated with it
- **RECEIVER** - software that consumes a message according to the protocol(s) associated with it (e.g., SOAP processors)

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. The Profile's scope is initially 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 by the Profile as an extensibility point, such a mechanism or parameter is outside the Profile's scope, and its use or non-use is not relevant to conformance.

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. 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.

This Profile's scope is defined by the referenced specifications in Appendix I, as refined by the extensibility points in Appendix II.

2.4 Claiming Conformance

Claims of conformance to the Profile can be made using the mechanisms described in the Profile Conformance Framework. Specifically, claims can be made using the following conformance attachment mechanisms, as long as the requirements in this profile associated with the listed targets have been met:

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

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

Generally, a deployed instance of a Web service (as specified by `wsdl:port` or `uddi:bindingTemplate`) is considered conformant if it produces only conformant artifacts, and is capable of consuming conformant artifacts, as appropriate. Note that this means that where multiple conformant artifacts are possible, a conformant service must be able to consume them all (e.g., while a sender might choose whether to encode XML in UTF-8 or UTF-16 when sending a message, a receiver must be capable of using either).

Note that conformance does not apply to a service as a whole; only ports are considered when determining conformance of instances. Therefore, the Profile places no constraints on `wsdl:service` definitions. In particular, they can contain multiple `wsdl:port` elements, each of which may or may not be conformant.

> **Editors' note:**There is still a need to turn the implied requirements in this section into actual Requirements.

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:

    - Header blocks - Header blocks are the fundamental extensibility mechanism in SOAP.
    - Processing order - The order of processing of a SOAP message's components (e.g., headers) is unspecified, and therefore may need to be negotiated out-of-band.
    - 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.
    - soap:actor values - The value of the soap:actor attribute is a private agreement between the parties to a Web service.
    - Fault details - the contents of a Fault's detail element are not prescribed by SOAP 1.1.
    - 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:
    - HTTP Authentication - HTTP authentication allows for extension schemes, arbitrary digest hash algorithms and parameters.
    - Unspecified Header Fields - HTTP allows arbitrary headers to occur in messages.
    - Expect-extensions - The Expect/Continue mechanism in HTTP allows for expect-extensions.
    - Content-Encoding - The set of content-codings allowed by HTTP is open-ended.
    - Transfer-Encoding - The set of transfer-encodings allowed by HTTP is open-ended.
    - Upgrade - HTTP allows a connection to change to an arbitrary protocol using the Upgrade header.
- 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 transmitting 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).*

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, these XML constructs are disallowed by section 3 of SOAP 1.1.

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

R1009 *An ENVELOPE MUST NOT contain Processing Instructions.* 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 elements are grandchildren 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. The `xsi:type` attribute is only needed where no such schema exists, that is where both sides are assuming that all exchanged items are "xsd:anyType".

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).*

3.2 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 that they do not understand targeted at them.

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.*

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.

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.*

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.*

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.*

3.3 SOAP Faults

3.3.1 SOAP Fault Structure

A fault is an envelope that has a single child element of the `soap:Body` element, that element being a `soap:Fault` element. The Profile restricts the content of the `soap:Fault` element to those elements explicitly described in SOAP 1.1.

R1000 *When an ENVELOPE contains a* `soap:Fault` *element, that 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>
```

3.3.2 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 contains a `soap:Fault` element its element children 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:fau
ltactor>
```

```
  <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>
```

3.3.3 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/".*

3.3.4 SOAP Fault Language

Faultstrings are human-readable indications of the n ature 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.

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

3.3.5 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 one of the fault codes defined in SOAP 1.1 or a namespace qualified fault code.*

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>
```

3.3.6 Identifying SOAP Faults

Some consumer implementations 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.

R1107 *A RECEIVER MUST interpret an envelope containing only a `soap:Fault` element as a 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 Header Syntax

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

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 and TCP Ports

SOAP is designed to take advantage of the HTTP infrastructure. However, there are some situations (e.g., involving proxies, firewalls and other intermediaries) where there may be harmful side effects. As a result, instances may find it advisable to use ports other than the default for HTTP (port 80).

R1110 *An INSTANCE MAY accept connections on TCP port 80 (HTTP).* c

There has been considerable debate within the W3C and IETF regarding the propriety of the use of port 80 for SOAP messages bound to HTTP. It has been concluded that this is an acceptable practice.

3.4.5 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.*

3.4.6 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.

3.4.7 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 payload of the HTTP request is not the proper media type (i.e., "text/xml", as required by the SOAP/HTTP binding), 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.*

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

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.

3.4.8 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.*

3.4.9 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)
- Web Services Description Language (WSDL) 1.1
  Extensibility points:

    - WSDL extensions - WSDL allows extension elements in certain places; use of such extensions requires out-of-band negotiation.
    - Relative URIs - WSDL does not adequately specify the use of relative URIs; their use may require further coordination; see XML Base for more information.
    - Validation mode - whether the parser used to read WSDL and XML Schema documents performs DTD validation or not.
    - Fetching of external resources - whether the parser used to read WSDL and XML Schema documents fetches external entities and DTDs.
- XML Schema Part 1: Structures
  Extensibility points:
    - Schema annotations - XML Schema allows for annotations, which may be used to convey additional information about data structures.
- XML Schema Part 2: Datatypes

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 *An INSTANCE MUST be described by a WSDL 1.1 service description, by a UDDI binding template, or both.*

"described," in this context, 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://schemas.xmlsoap.org/wsdl/2003-02-11.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://schemas.xmlsoap.org/wsdl/soap/2003-02-11.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.*

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 *A DESCRIPTION MUST NOT use the XML Schema "import" statement to import a Schema from 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/definiti
ons"
   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/definiti
ons">
      <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 *In a DESCRIPTION the value of the `location` attribute of a `wsdl:import` element SHOULD be treated as a hint.* C

This means that WSDL processor may, but need not, retrieve a WSDL description from the URI specified in the `location` attribute on a `wsdl:import` element because a WSDL processor may have other ways of locating a WSDL description for a given namespace. For example, it may already have a cached or built-in representation, or it may retrieve a representation from a metadata repository or UDDI server.

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/definiti
ons">

     <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 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

4.2.8 Acceptable WSDL Character Encodings

The Profile consistently requires either UTF-8 or UTF-16 encoding for both SOAP and WSDL (see also R1012).

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

4.2.9 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.*

4.2.10 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.

R2020 *The `wsdl:documentation` element MAY occur as a child of the `wsdl:import` element in a DESCRIPTION.* WSDL12

R2021 *The `wsdl:documentation` element MAY occur as a child of the `wsdl:part` element in a DESCRIPTION.* WSDL12

R2024 *The `wsdl:documentation` element MAY occur as a first child of the `wsdl:definitions` element in a DESCRIPTION.* WSDL12

4.2.11 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 an element in the WSDL namespace in a description, a consumer encounters a WSDL extension element amongst its element children, 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 of that element in the WSDL namespace.*

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.

The following elements are extensible via attributes only:

- `wsdl:import`
- `wsdl:part`
- `wsdl:portType`
- `wsdl:input` (in portType operation)
- `wsdl:output` (in portType operation)
- `wsdl:fault` (in portType operation)

The following elements are extensible via elements as well as attributes:

- `wsdl:definitions`
- `wsdl:types`
- `wsdl:message`
- `wsdl:operation`
- `wsdl:binding`
- `wsdl:input` (in binding operation)
- `wsdl:output` (in binding operation)
- `wsdl:fault` (in binding operation)
- `wsdl:service`
- `wsdl:port`

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. Similar to XML Schema, namespaces not referenced directly within the WSDL file (through the `targetNamespace` attribute on `xsd:schema`, or through the namespace attribute on `xsd:import`) are available for use in QName reference. QName references to namespaces that are only defined through a nested import are not allowed.

R2101 *A DESCRIPTION MUST NOT use QName references to elements 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, array declarations MUST NOT extend or restrict the* `soapenc:Array` *type.*

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

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

R2113 *An ENVELOPE containing serialized arrays 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.* WSDL12

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 wire format. 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.

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` each of whose `soapbind:body` descendant elements specifies the `use` attribute with the value "literal" and each of which either:

1. Specifies the style attribute with the value "rpc"; or
2. Is the child of a `soapbind:binding` element which specifies the `style` attribute with the value "rpc", and does not itself have the `style` attribute specified.

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` each of whose `soapbind:body` descendent elements specifies the `use` attribute with the value "literal" and each of which either:

1. Specifies the `style` attribute with the value "document"; or
2. Is the child of a `soapbind:binding` element which specifies the `style` attribute with the value "document", and does not itself have the `style` attribute specified; or
3. Is the child of a `soapbind:binding` element which does not have the `style` attribute specified, and does not itself have the `style` attribute specified.

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.*

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).*

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 wire representation of that part 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".

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 Unbound portType Element Contents

WSDL 1.1 is not explicit about whether it is permissible for a `wsdl:binding` to leave the binding for portions of the content defined by a `wsdl:portType` unspecified.

R2209 *A `wsdl:binding` in a DESCRIPTION SHOULD bind every `wsdl:part` of a `wsdl:message` in the `wsdl:portType` to which it refers to one of `soapbind:body`, `soapbind:header`, `soapbind:fault` or `soapbind:headerfault`.*

A portType defines an abstract contract with a named set of operations and associated abstract messages. Although not disallowed, it is expected that every part of the abstract input, output and fault messages specified in a portType is bound to `soapbind:body` or `soapbind:header` (and so forth) as appropriate when using the SOAP binding as defined in WSDL 1.1 Section 3.

4.4.4 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:part`s in the `wsdl:message` that describes it.*

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:portType` should be constructed.

R2305 *A `wsdl:portType` 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 use 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 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.*

4.7.6 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.*

4.7.7 Wire Signatures for Operations

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 wire signature.

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

The Profile defines the "wire signature" of an operation in a `wsdl:binding` to be the fully qualified name of the child element of the `soap:Body` of the SOAP input message it describes. For the case of an empty `soap:Body` this name is an empty string.

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.

4.7.8 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.*

4.7.9 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 represented 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.*

4.7.10 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.

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.

4.7.11 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.

4.7.12 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

4.7.13 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.

4.7.14 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 `wsdl: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 `wsdl:headerfault` element in the corresponding WSDL description.*

4.7.15 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>
```

4.7.16 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.*

4.7.17 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".*

R2728 *A `wsdl:binding` in a DESCRIPTION that omits the `use` attribute on a contained `soapbind:fault` element MUST be interpreted as though `use`="literal" had been specified.* C

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".

4.7.18 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 Namespace for 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.*

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 children of part accessor elements for the parameters and the return value with the targetNamespace in which their 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 message="bar:BarMsg">
       <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 both 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, and defines extensibility points within them;

- 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

UDDI represents Web service instances as `uddi:bindingTemplate` elements. The `uddi:bindingTemplate` plays a role that is the rough analog of the `wsdl:port`, but provides options that are not expressible in WSDL. To keep the WSDL description of an instance and its UDDI description consistent, the Profile places the following constraints on how `uddi:bindingTemplate` elements may be constructed.

WSDL's `soapbind:address` element requires the network address of the instance to be directly specified. In contrast, UDDI V2 provides two alternatives for specifying the network address of instances it represents. One, the `uddi:accessPoint`, mirrors the WSDL mechanism by directly specifying the address. The other, the `uddi:hostingRedirector`, provides a Web service-based indirection mechanism for resolving the address, and is inconsistent with the WSDL mechanism.

R3100 *REGDATA of type `uddi:bindingTemplate` representing a conformant INSTANCE MUST contain the `uddi:accessPoint` element.*

For example,

INCORRECT:

```
<bindingTemplate bindingKey="...">
   <description xml:lang="EN">BarSOAPPort</description>
   <hostingRedirector bindingKey="..."/>
   <tModelInstanceDetails>
      ...
   </tModelInstanceDetails>
</bindingTemplate>
```

CORRECT:

```
<bindingTemplate bindingKey="...">
   <description xml:lang="EN">BarSOAPPort</description>

<accessPoint>http://example.org/myBarSOAPPort</accessPo
int>
   <tModelInstanceDetails>
      ...
   </tModelInstanceDetails>
```

```
</bindingTemplate>
```

5.2 tModels

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

- UDDI Version 2.03 Data Structure Reference, Section 8

UDDI represents Web service types as `uddi:tModel` elements. (See UDDI Data Structures section 8.1.1.) These may, but need not, point (using a URI) to the document that contains the actual description. Further, UDDI is agnostic with respect to the mechanisms used to describe Web service types. The Profile cannot be agnostic about this because interoperation is very much complicated if Web service types do not have descriptions or if the descriptions can take arbitrary forms.

The UDDI API Specification, appendix I.1.2.1.1 allows but does not require `uddi:tModel` elements that use WSDL to describe the Web service type they represent to state that they use WSDL as the description language. Not doing so leads to interoperability problems because it is then ambiguous what description language is being used.

Therefore the Profile places the following constraints on how `uddi:tModel` elements that describe Web service types may be constructed:

The Profile chooses WSDL as the description language because it is by far the most widely used such language.

R3002 *REGDATA of type `uddi:tModel` representing a conformant Web service type MUST use WSDL as the description language.*

To specify that conformant Web service types use WSDL, the Profile adopts the UDDI categorization for making this assertion.

R3003 *REGDATA of type `uddi:tModel` representing a conformant Web service type MUST be categorized using the uddi:types taxonomy and a categorization of "wsdlSpec".*

For the `uddi:overviewURL` in a `uddi:tModel` to resolve to a `wsdl:binding`, the Profile must adopt a convention for distinguishing among multiple `wsdl:binding`s in a WSDL document. The UDDI Best Practice for Using

WSDL in a UDDI Registry specifies the most widely recognized such convention.

R3010 *REGDATA of type* `uddi:tModel` *representing a conformant Web service type MUST follow V1.08 of the UDDI Best Practice for Using WSDL in a UDDI Registry.*

It would be inconsistent if the `wsdl:binding` that is referenced by the `uddi:tModel` does not conform to the Profile.

R3011 *The wsdl:binding that is referenced by REGDATA of type* `uddi:tModel` *MUST itself conform to the Profile.*

6. Security

As is true of all network-oriented information technologies, the subject of security is a crucial one for Web services. For Web services, as for other information technologies, security consists of understanding the potential threats an attacker may mount and applying operational, physical, and technological countermeasures to reduce the risk of a successful attack to an acceptable level. Because an "acceptable level of risk" varies hugely depending on the application, and because costs of implementing countermeasures is also highly variable, there can be no universal "right answer" for securing Web services. Choosing the absolutely correct balance of countermeasures and acceptable risk can only be done on a case by case basis.

That said, there *are* common patterns of countermeasures that experience shows reduce the risks to acceptable levels for many Web services. The Profile adopts, but does not mandate use of, the most widely used of these: HTTP secured with either TLS 1.0 or SSL 3.0 (HTTPS). That is, conformant Web services may use HTTPS; they may also use other countermeasure technologies or none at all.

HTTPS is widely regarded as a mature standard for encrypted transport connections to provide a basic level of confidentiality. HTTPS thus forms the first and simplest means of achieving some basic security features that are required by many real-world Web service applications. HTTPS may also be used to provide client authentication through the use of client-side certificates.

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

- RFC2818: HTTP Over TLS
- RFC2246: The TLS Protocol Version 1.0
  Extensibility points:
  - TLS Cyphersuite - TLS allows for the use of arbitrary encryption algorithms.
  - TLS Extensions - TLS allows for extensions during the handshake phase.
- The SSL Protocol Version 3.0
  Extensibility points:
  - SSL Cyphersuite - SSL allows for the use of arbitrary encryption algorithms.
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
  Extensibility points:
  - Certificate Authority - The choice of the Certificate Authority is a private agreement between parties.
  - Certificate Extensions - X509 allows for arbitrary certificate extensions.

6.1 Use of HTTPS

HTTPS is such a useful, widely understood basic security mechanism that the Profile needs to allow it.

R5000 *An INSTANCE MAY require the use of HTTPS.*

R5001 *If an INSTANCE requires the use of HTTPS, the location attribute of the* `soapbind:address` *element in its* `wsdl:port` *description MUST be a URI whose scheme is "https"; otherwise it MUST be a URI whose scheme is "http".*

Simple HTTPS provides authentication of the Web service instance by the consumer but not authentication of the consumer by the instance. For many instances this leaves the risk too high to permit interoperation. Including the mutual authentication facility of HTTPS in the Profile permits instances to use the countermeasure of authenticating the consumer. In cases in which authentication of the instance by the consumer is insufficient, this often reduces the risk sufficiently to permit interoperation.

R5010 *An INSTANCE MAY require the use of HTTPS with mutual authentication.*

Appendix I: Referenced Specifications

The following specifications' requirements are incorporated into the Profile by reference, except where superseded by the Profile:

- Simple Object Access Protocol (SOAP) 1.1
- RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
- RFC2965: HTTP State Management Mechanism
- Extensible Markup Language (XML) 1.0 (Second Edition)
- Web Services Description Language (WSDL) 1.1
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- 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
- RFC2818: HTTP Over TLS
- RFC2246: The TLS Protocol Version 1.0
- The SSL Protocol Version 3.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile

Appendix II: Extensibility Points

This section identifies extensibility points, as defined in "Scope of the Profile," for the Profile's component specifications.

These mechanisms are out of the scope of the Profile; their use may affect interoperability, and may require private agreement between the parties to a Web service.

In Simple Object Access Protocol (SOAP) 1.1:

- **Header blocks** - Header blocks are the fundamental extensibility mechanism in SOAP.
- **Processing order** - The order of processing of a SOAP message's components (e.g., headers) is unspecified, and therefore may need to be negotiated out-of-band.
- **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.
- **soap:actor values** - The value of the soap:actor attribute is a private agreement between the parties to a Web service.
- **Fault details** - the contents of a Fault's detail element are not prescribed by SOAP 1.1.
- **Envelope serialization** - The Profile does not constrain some aspects of how the envelope is serialized into the message.

In RFC2616: Hypertext Transfer Protocol -- HTTP/1.1:

- **HTTP Authentication** - HTTP authentication allows for extension schemes, arbitrary digest hash algorithms and parameters.
- **Unspecified Header Fields** - HTTP allows arbitrary headers to occur in messages.
- **Expect-extensions** - The Expect/Continue mechanism in HTTP allows for expect-extensions.
- **Content-Encoding** - The set of content-codings allowed by HTTP is open-ended.
- **Transfer-Encoding** - The set of transfer-encodings allowed by HTTP is open-ended.
- **Upgrade** - HTTP allows a connection to change to an arbitrary protocol using the Upgrade header.

In Web Services Description Language (WSDL) 1.1:

- **WSDL extensions** - WSDL allows extension elements in certain places; use of such extensions requires out-of-band negotiation.
- **Relative URIs** - WSDL does not adequately specify the use of relative URIs; their use may require further coordination; see XML Base for more information.
- **Validation mode** - whether the parser used to read WSDL and XML Schema documents performs DTD validation or not.
- **Fetching of external resources** - whether the parser used to read WSDL and XML Schema documents fetches external entities and DTDs.

In XML Schema Part 1: Structures:

- **Schema annotations** - XML Schema allows for annotations, which may be used to convey additional information about data structures.

In RFC2246: The TLS Protocol Version 1.0:

- **TLS Cyphersuite** - TLS allows for the use of arbitrary encryption algorithms.
- **TLS Extensions** - TLS allows for extensions during the handshake phase.

In The SSL Protocol Version 3.0:

- **SSL Cyphersuite** - SSL allows for the use of arbitrary encryption algorithms.

In RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile:

- **Certificate Authority** - The choice of the Certificate Authority is a private agreement between parties.
- **Certificate Extensions** - X509 allows for arbitrary certificate extensions.

Appendix III: Acknowledgements

This Profile is the work of the WS-I Basic Profile Working Group, whose members have included:

Mark Allerton (Crystal Decisions Corporation), George Arriola (Talking Blocks, Inc.), Keith Ballinger (Microsoft Corporation), Ilya Beyer (KANA), Rich Bonneau (IONA Technologies), Don Box (Microsoft Corporation), Andrew Brown (Verisign), Heidi Buelow (Quovadx), David Burdett (Commerce One, Inc.), Luis Felipe Cabrera (Microsoft Corporation), Maud Cahuzac (France Telecom), Bhaskar Chakrabarti (United Airlines), Chia Chao (IONA Technologies), Martin Chapman (Oracle Corporation), Richard Chin (Avinon), Roberto Chinnici (Sun Microsystems), Sergio Compean (Avanade, Inc.), Tim Cooke (Onyx Software), Ugo Corda (SeeBeyond Tech), Paul Cotton (Microsoft Corporation), Joseph Curran (Accenture), Ayse Dilber (AT&T), Dave Ehnebuske (IBM), Mark Ericson (Mindreef Inc.), Colleen Evans (Sonic Software), Tim Ewald (Microsoft Corporation), Chuck Fay (FileNet Corporation), Chris Ferris (IBM), Daniel Foody (Actional Corporation), Toru Fujii (NTT), Satoru Fujita (NEC Corporation), Shishir Garg (France Telecom), Yaron Goland (BEA Systems), Hans Granqvist (Verisign), Martin Gudgin (Microsoft Corporation), Marc Hadley (Sun Microsystems), Bob Hall (Unisys Corporation), Scott Hanselman (Corillian), Muir Harding (Autodesk, Inc.), Loren Hart (Verisign), Harry Holstrom (Accenture), Larry Hsiung (Quovadx), Hemant Jain (Tata Consultancy), Steve Jenisch (SAS Institute), Erik Johnson (Epicor Software Corporation), Bill Jones (Oracle Corporation), Menno Jonkers (Tryllian BV), Anish Karmarkar (Oracle Corporation), Takahiro Kawamura (Toshiba), Bhushan Khanal (WRQ, Inc.), Sunil Kunisetty (Oracle Corporation), Canyang Kevin Liu (SAP AG), Jonathan Marsh (Microsoft Corporation), David Meyer (Plumtree Software, Inc.), Jeff Mischkinsky (Oracle Corporation), Tom Moog (Sarvega Inc.), Gilles Mousseau (Hummingbird Ltd.), Richard Nikula (BMC Software, Inc.), Eisaku Nishiyama (Hitachi, Ltd.), Mark Nottingham (BEA Systems), David Orchard (BEA Systems), Jesse Pangburn (Quovadx), TJ Pannu (ContentGuard, Inc.), Eduardo Pelegri-Llopart (Sun Microsystems), Vijay Rajan (Novell), Eric Rajkovic (Oracle Corporation), Graeme Riddell (Bowstreet), Claus von Riegen (SAP AG), Tom Rutt (Fujitsu Limited), Roger Sanborn (Crystal Decisions

Corporation), Krishna Sankar (Cisco Systems, Inc.), Don Schricker (Micro Focus), Dave Seidel (Mindreef Inc.), Akira Shimaya (NTT), Yasser Shohoud (Microsoft Corporation), David Smiley (Mercator Software, Inc.), Seumas Soltysik (IONA Technologies), Joseph Stanko (Plumtree Software, Inc.), Keith Stobie (Microsoft Corporation), Yasuo Takemoto (NTT), Nobuyoshi Tanaka (NEC Corporation), Jorgen Thelin (Cape Clear Software), Sameer Vaidya (Talking Blocks, Inc.), William Vambenepe (Hewlett-Packard), Rick Weil (Eastman Kodak Company), Scott Werden (WRQ, Inc.), Ajamu Wesley (IBM), Shannon Wheatley (Kinzan, Inc.), Ian White (Micro Focus), Sue Worthman (Tryllian BV), Prasad Yendluri (webMethods Inc.).