Basic Profile Version 1.0

Working Group Draft

Date: 2003/01/24 09:47:33

This revision:
http://www.ws-i.org/Profiles/Basic/2003-01/BasicProfile-1.0-WGD.html

Editors:
Keith Ballinger, Microsoft
David Ehnebuske, IBM
Martin Gudgin, Microsoft
Mark Nottingham, BEA Systems
Prasad Yendluri , webMethods
Administrative contact:
secretary@ws-i.org

Abstract

This document defines the WS-I Basic Profile, 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 supercede this document.

The Working Group believes this Working Group Draft to be substantively complete with regards to the technical refinements, clarifications and constraints specified. The Working

Group will be publishing another Working Group Draft reflecting the remaining issue resolutions, additional rationale text, and editorial changes within the next month, and intends to pursue an aggressive schedule for completion of its work. Therefore, the Working Group invites all parties interested in contributing to the review and feedback process of the Basic Profile v1.0 to do so based on this version so that any technical concerns expressed can be considered by the Working Group before it concludes its formal review period.

Notice

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.

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

Table of Contents

1. Introduction

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

Section 1 introduces the profile, its scope, and the philosophy that it takes to interoperability.

Section 2, "Profile Conformance," explains what it means to be conformant to the Basic Profile. Each subsequent section addresses a component specification of the Profile, and consists of two parts; an overview of the approach to the specification taken, followed by subsections which address individual parts of the component specification.

1.1 Guiding Principles

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

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

Application semantics
: Although communication of application semantics can be facilitated by the technologies that comprise the Profile, assuring the common understanding of those semantics is out of scope.

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

Strength of requirements
: The Profile makes strong requirements (e.g., MUST, MUST NOT) wherever feasible; if there are legitimate cases where such a requirement cannot be met, conditional requirements (e.g., SHOULD, SHOULD NOT) are used. Optionally is avoided as much as possible.

Restriction vs. relaxation
: When amending the requirements of referenced specifications, the Profile may restrict their requirements, but does not relax them (i.e., change a MUST to a MAY). Optional and conditional requirements introduce ambiguity and mismatches between implementations.

Multiple mechanisms
: If a referenced specification allows multiple mechanisms to be used interchangeably, the Profile selects those that are well-understood, widely implemented and useful.

Extraneous or underspecified mechanisms and extensions introduce complexity and therefore reduce interoperability.

Future compatibility

When possible, the Profile aligns its requirements with in-progress revisions to the specifications it references (e.g., SOAP 1.2, WSDL 1.2). This aids implementers by enabling a graceful transition, and assures that WS-I does not 'fork' from these efforts. When the Profile cannot address an issue in a specification it references, this information is communicated to the appropriate body to assure their consideration.

Compatibility with deployed services

Backwards compatibility with deployed Web services is not a goal for the Profile, but due consideration is given to it; the Profile does not introduce a change to existing specifications unless there are specific interoperability issues.

Focus on interoperability

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

Conformance targets

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

Lower-layer interoperability

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

1.2 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 section 3) are presented in the following manner:

Rnnnn*Statement text here.*

where "nnnn" is replaced by the statement number. Each statement will contain 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, 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, currently under development). Note that because such work is not complete, the specification which 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/

2. Profile Conformance

Conformance to the Basic Profile is defined by adherence to the specifications on which the profile is based (as outlined in the remainder of the document), subject to the refinements, interpretations, and clarifications set forth.

To allow the description of conformance in different contexts, the profile defines a number of *conformance targets*, allowing the conformance testing and certification of artifacts (such as SOAP messages and WSDL descriptions), Web services themselves, and software that is used in conjunction with a conformant Web Service.

The criteria for conformance is defined by *requirement statements*, which are associated with conformance targets (denoted with capital letters, e.g., MESSAGE) and use requirement levels (using RFC2119 language, e.g., MUST) to indicate the nature of the requirement. Requirement statements are individually identified (e.g., r999) for convenience. Additional text may be included to illuminate the requirements (e.g., rationale and examples); however, requirement statements alone should be considered in determining conformance.

The sections below describe this profile's conformance targets, from the basic artifacts (upon which requirements are directly placed) to the conformance of services and software, which is derived from these artifacts and additional requirements.

2.1 Conformance of Artifacts

The most basic level of conformance is that of an artifact. This profile makes requirement statements about three kinds of artifacts;

- **MESSAGE** - protocol elements that are exchanged, usually over a network, to effect a Web service (i.e., 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 (i.e., WSDL descriptions)
- **REGDATA** - statements about Web services that are used to discover their capabilities (e.g., UDDI tModels)

An instance of an artifact is considered conformant when all of the requirements associated with it are met.

2.2 Conformance of Services and Consumers

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).

Conformant Web service instances must comply with all requirement statements associated with **INSTANCE**.

Similarly, a consumer of a service instance is considered conformant if it produces only conformant artifacts and is capable of consuming conformant artifacts, as appropriate.

Conformant consumers must comply with all requirements statements associated with **CONSUMER**.

Both conformant Web service instances and consumers must comply, as appropriate, with all of the requirement statements associated with:

- **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)

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

Types of Web services (as specified by wsdl:binding and wsdl:portType) are considered conformant if, when deployed with due consideration, they produce conformant instances.

Additionally, 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.

2.3 Conformance Annotation

To allow services to advertise conformance to the Profile, *conformance claims* regarding instances can be placed at the appropriate place in a service's WSDL description. Claims can be associated with a particular element (e.g., portType) to scope them to that construct.

> R0002 *DESCRIPTIONs MAY contain conformance claims regarding instances, as specified in the conformance claim schema.*

> R0003 *DESCRIPTIONs' conformance claims MUST be children of the documentation element of each of the wsdl elements: port, binding, portType, operation and message.*

A conformance claim on an element means that the element (and the instance it represents, in the case of a port) is conformant to the profile it claims to obey (as relevant to the type of element).

A conformance claim on an element also implies that the same claim is made for all the elements that it uses, based on the following transitivity rules, applied recursively:

- A claim on a port is inherited by the referenced binding
- A claim on a binding is inherited by the referenced portType
- A claim on a portType is inherited by the referenced operations
- A claim on an operation is inherited by the referenced messages

The conformance claim schema is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://ws-i.org/schemas/conformanceClaim/"
          xmlns:tns="http://ws-i.org/schemas/conformanceClaim/"
```

```
           elementFormDefault="qualified"
           attributeFormDefault="unqualified" >
    <xs:element name="Claim" >
        <xs:complexType>
            <xs:sequence>
                <xs:any namespace="##any" processContents="lax"/>
            </xs:sequence>
            <xs:attribute name="conformsTo" type="xs:anyURI"/>
            <xs:anyAttribute namespace="##any" processContents="lax"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

For example,

CORRECT:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
                  xmlns:tns="http://example.org/myservice"
                  targetNamespace="http://example.org/myservice">
    <wsdl:portType name="MyPortType">
        ...
    </wsdl:portType>
    <wsdl:binding name="MyBinding" portType="MyPortType">
        ...
    </wsdl:binding>
    <wsdl:service name="MyService">
        <wsdl:port name="MyPort" binding="tns:MyBinding">
            <wsdl:documentation>
              <wsi:Claim conformsTo="http://ws-
i.org/profiles/basic1.0/"/>
            </wsdl:documentation>
            <soap:address
location="http://example.org/myservice/myport"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Editors' note: The editors have an outstanding action item to obtain a URI for use in conjunction with conformance claims.

3. Messaging

This portion of the profile incorporates the following specifications by reference;

- Simple Object Access Protocol (SOAP) 1.1 .
- Extensible Markup Language (XML) 1.0 (Second Edition).
- RFC2616: Hypertext Transfer Protocol -- HTTP/1.1.
- RFC2965: HTTP State Management Mechanism.

3.1 XML Representation of SOAP Messages

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- SOAP 1.1, Section 4.

SOAP 1.1 defines an XML-based structure for transmitting messages. This profile mandates the use of that structure, and places the following constraints on its use:

The XML specification allows UTF-8 encoding to include a BOM; therefore, receivers of messages must be prepared to accept them. The BOM is mandatory for XML encoded as UTF-16.

R4001 *A RECEIVER MUST accept messages that include the Unicode Byte Order Mark (BOM).*c

For interoperability the content of the `soap:Fault` element is restricted to those elements explicitly described in the SOAP 1.1 specification.

R1000 *When a MESSAGE 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>There were <b>lots</b> of elements in the message
   that I did not understand
   </detail>
</soap:Fault>
```

The children of the `soap:Fault` element are local to that element and do not need to be namespace qualified.

R1001 *When a MESSAGE 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:faultactor>
  <soap:detail>There were <b>lots</b> of elements in the message
  that I did not understand
  </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>There were <b>lots</b> of elements in the message
  that I did not understand
  </detail>
```

```
</soap:Fault>
```

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 fault messages 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 fault messages that have any number of qualified attributes, including zero, appearing on the `detail` element. The namespace of such attributes can be anything other than "http://schemas.xmlsoap.org/soap/envelope/".*

To help internationalization the language of faultstrings may be indicated.

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

Custom fault codes must not appear inside the `faultcode` element; for interoperability, a fixed set of fault codes is needed.

R1004 *When a MESSAGE contains a `faultcode` element the content of that element MUST be one of the fault codes defined in the SOAP 1.1 specification.*

For example,

INCORRECT:

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

For interoperability, literal XML is preferred.

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

R1006 *A MESSAGE MUST NOT contain `soap:encodingStyle` attributes on any element which is a child of `soap:Body`.*

R1007 *A MESSAGE MUST NOT contain `soap:encodingStyle` attributes on any elements which are grandchildren of `soap:Body`.*

For interoperability, security and ease of processing these XML constructs are disallowed.

R1008 *A MESSAGE MUST NOT contain a Document Type Declaration.* C

R1009 *A MESSAGE MUST NOT contain Processing Instructions.* C

Presence or absence of such a declaration does not affect interoperability. Certain implementations might always precede their XML serialization with the XML declaration.

R1010 *A RECEVIER MUST accept messages that contain an XML Declaration.* C

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

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

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>
```

All XML Processors must support UTF-8 and UTF-16, per the XML 1.0 specification so requiring these encodings places no burden on implementations and aids interoperability.

> R1012 *A MESSAGE MUST be serialized as either UTF-8 or UTF-16.*

The `soap:mustUnderstand` attribute has a type of xsd:boolean which allows all four lexical forms but for compatibility only two are allowed.

> R1013 *MESSAGEs containing a `soap:mustUnderstand` attribute MUST only use the lexical forms 0 and 1.* c

The interpretation of unqualified element names is ambiguous, therefore qualified names must be used.

> R1014 *The children of the `soap:Body` element in a MESSAGE MUST be namespace qualified.*

SOAP 1.1 only stated that the message be discarded in such cases. For interoperability faults must be generated instead.

> R1015 *A RECEIVER MUST generate a fault if they encounter a message whose document element has a local name of "Envelope" but a namespace name which is not "http://schemas.xmlsoap.org/soap/envelope/".*

In many cases senders and receivers will share some form of type information related to the messages being exchanged. xsi:type is only needed where no such schema exists, that is where both sides are assuming that all exchanged items are xs:anyType.

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

3.2 The SOAP Processing Model

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- SOAP 1.1, Section 2.

SOAP 1.1 defines a message exchange model for processing of messages. This profile places the following constraints on that model:

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

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

`soap:actor` is used to target header blocks at intermediaries. The SOAP specification has nothing to say about its value.

> R1026 *The value of the `soap:actor` attribute in a MESSAGE is a private agreement between the sender and the receiver of the header carrying the attribute.* C

> **Editors' note:**This statement isn't really a requirement; it might become a Best Practice.

Requiring that receivers generate faults ensures that mandatory headers are not silently and erroneously ignored.

> R1027 *A RECEIVER MUST generate a soap:MustUnderstand fault when a message contains a mandatory header targeted at the receiver (via soap:actor) that the receiver does not understand. A mandatory header is one which carries a `soap:mustUnderstand` attribute with the value 1.*

These requirements ensure that, when a Fault is generated, no further processing will be done on the message, a Fault message will be transmitted to the sender of the request message in request-response cases and some application level error will be flagged to the user.

> R1028 *Upon generating a SOAP Fault a RECEIVER MUST NOT effect any further processing of a SOAP message beyond that which is necessary to handle the generated SOAP Fault.*

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

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

3.3 Using SOAP in HTTP

This portion of the profile modifies and refers to the following specifications (or sections thereof);

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

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

HTTP/1.1 has several performance advantages and is more clearly specified, in comparison to HTTP/1.0. 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.

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

Some consumer implementations use only the HTTP status code to determine the presence of a SOAP 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 SOAP messages containing only a `soap:Fault` element as a Fault.*

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

R1108 *A MESSAGE MUST NOT use the HTTP Extension Framework [RFC2774].*

Interoperability testing has demonstrated that requiring the value of the `SOAPAction` HTTP Header Field to be quoted increases interoperability of implementations. Even though HTTP allows for the value of HTTP Header Fields 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.

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

R1117 *A MESSAGE MUST contain a `SOAPAction` HTTP header field with a quoted value equal to the value of the `soapbind:operation/@soapAction` attribute, if present in the DESCRIPTION.* C

R1118 *A MESSAGE MUST contain a `SOAPAction` HTTP Header Field with a quoted empty string if the `soapbind:operation/@soapAction` attribute is either not present in the DESCRIPTION, or is present and contains an empty string as its value.* C

R1119 *A RECEIVER MAY respond with a Fault if the value of the* `SOAPAction` *HTTP Header Field is not quoted.* C

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: ""
```

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 use other ports.

R1110 *An INSTANCE MAY use TCP port 80 (HTTP).* C

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 messages 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 for responses that indicate a successful outcome of a request.*

R1111 *An INSTANCE SHOULD use a "200 OK" HTTP status code for responses that contain a SOAP message that is not a SOAP fault.*

R1112 *An INSTANCE SHOULD use either a "200 OK" or "202 Accepted" HTTP status code for responses that indicate successful HTTP outcome of a request but do not contain a SOAP message.*

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" as the correct status code for redirection; for more information, see the 3xx status code descriptions in RFC2616.

R1130 *An INSTANCE MUST use HTTP status code "307 Temporary Redirect" 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 come to be configured to automatically follow redirections.

HTTP uses the 4xx series of status codes to indicate failure due to a client error. Although there are a number of situations which may result in one of these codes, the profile highlights those when the payload of the HTTP request is not the proper media type, and when the anticipated method is not used.

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

R1113 *An INSTANCE SHOULD use a "400 Bad Request" HTTP status code if the request message is invalid (HTTP request malformed, XML not well formed, ...).*

R1114 *An INSTANCE SHOULD use a "405 Method not Allowed" HTTP status code if the request method was not "POST".*

R1115 *An INSTANCE SHOULD use a "415 Unsupported Media Type" HTTP status code if the Content-Type HTTP request header did not have a value of "text/xml".*

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

R1116 *An INSTANCE MUST use a 5xx HTTP status code for responses that indicate an unsuccessful outcome of a well formed request.*

R1126 *An INSTANCE MUST use a "500 Internal Server Error" HTTP status code if the response message contains a SOAP Fault.*

The HTTP State Management Mechanism, or "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 SOAP Envelope, are not accommodated by either the SOAP or WSDL specifications. 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 on the consumer side - 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 a set of endpoints operating on messages.

This portion of the profile incorporates the following specifications by reference;

- Web Services Description Language (WSDL) 1.1.
- XML Schema Part 1: Structures.
- XML Schema Part 2: Datatypes.

4.1 Document Structure

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.1.

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

Some of the examples in WSDL 1.1 specification incorrectly show WSDL import statement being used to import XML Schema definitions. To promote interoperability the profile keeps the import mechanisms consistent and confined to their respective domains; the wsdl related mechanism in the "wsdl" domain and the schema related mechanisms in "schema" domain where the normal rules from the schema specification can be applied consistently.

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 only use the XML Schema "import" statement within the xsd:schema element of the types section.*

R2004 *A DESCRIPTION MUST NOT use the XML Schema "import" statement to import a Schema definition embedded in line within another WSDL description.*

For example,

INCORRECT:

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

   <import namespace="http://example.com/stockquote/schemas"
         location="http://example.com/stockquote/stockquote.xsd"/>

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

CORRECT:

```
<definitions name="StockQuote"
   targetNamespace="http://example.com/stockquote/definitions"
   xmlns:xsd1="http://example.com/stockquote/schemas"
```

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

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

       <xsd:import namespace="http://example.com/stockquote/schemas"
         schemaLocation="http://example.com/stockquote/stockquote.xsd"/>
         ...
         ...
     </xsd:schema>
   </types>

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

CORRECT:

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

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

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

The XML specification allows UTF-8 encoding to include a BOM; therefore, description processors must be prepared to accept them.

R4002 *DESCRIPTIONs MAY include the Unicode Byte Order Mark (BOM)*

The profile consistently requires either UTF-8 or UTF-16 encoding for both SOAP and WSDL. See also R1012.

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

Namespace coercion on wsdl:import is disallowed by the Basic Profile, as it has been found to be undesirable from interoperability perspective.

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.*

Eliminate confusion about whether the `location` attribute of the `wsdl:import` statement is required and what its content is required to be.

R2007 *A DESCRIPTION MUST specify a valid and non-null value for the `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.

A WSDL processor may not need to retrieve a WSDL description from the URI specified in the location attribute on a wsdl:import element. A WSDL processor may have other ways of locating a WSDL description for a given namespace, including but not limited to; already having a cached representation, having a built-in representation, retrieving a representation from a metadata repository or UDDI server.

R2008 *In a DESCRIPTION the value of the `location` attribute of a `wsdl:import` element SHOULD be treated as a hint.* C

Eliminate inconsistency between WSDL 1.1 schema and the WSDL 1.1 specification in this area.

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

Eliminate confusion created by example 3 in section 3.1 of the WSDL 1.1 specification and also align with the W3C WSDL WG resolution on this.

R2022 *In a DESCRIPTION the `wsdl:import` element(s), when present, MUST occur either prior to any other child elements under the `wsdl:definitions` element, if no `wsdl:documentation` element is present; or immediately following the `wsdl:documentation` element if present.* WSDL12

R2023 *In a DESCRIPTION the `wsdl:types` element MUST occur either as the first child of the `wsdl:definitions` element if no `wsdl:documentation` or `wsdl:import` elements are present; or immediately following the `wsdl:documentation` and/or `wsdl:import` elements if present.* WSDL12

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.xsd"
               xmlns="http://www.w3.org/2001/XMLSchema">
           .......
      </schema>
   </types>
</definitions>
```

CORRECT:

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

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

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

  <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
         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>
```

CORRECT:

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

  <types>
     <schema targetNamespace="http://example.com/stockquote.xsd"
          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 Types

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- 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. This 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:

The WSDL specification is not explicit that WSDL documents may only refer, using a QName, to a name that is in a namespace that has either been imported or defined in the referring document.

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

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.*

The recommendations in section 2.2 of the WSDL 1.1 specification for declaration of Array types are prohibited by the Basic Profile, as they are based on SOAP 1.1 encoding scheme.

R2110 *In a DESCRIPTION `array` declarations MUST NOT extend 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 *MESSAGEs with serialized form of arrays MUST NOT include the `soapenc:arrayType` attribute.*

For example,

INCORRECT:

```
<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>


In a SOAP message this would serialize as below (omitting namespace
declarations for clarity):


<MyArray2 soapenc:arrayType="tns:MyArray2Type[]" >

  <x>abcd</x>

  <x>efgh</x>

</MyArray2>
```

CORRECT:

```
<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>


In a SOAP message this would serialize as below (omitting namespace
declarations for clarity):
<MyArray1>
  <x>abcd</x>
  <x>efgh</x>
</MyArray1>
```

4.3 Messages

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.3.

In WSDL 1.1 message elements are used to represent abstract definitions of the data being transmitted and binding elements to define how the abstract definitions are bound to a specific wire format.

In discussing this the following definitions are useful.

An "rpc-literal binding" is a `wsdl:binding` that contains a `soapbind:binding` element whose `style` attribute has the value "rpc" and all of whose `soapbind:body` elements specify the `use`="literal" attribute.

A "document-literal binding" is a `wsdl:binding` that contains a `soapbind:binding` element whose `style` attribute is either omitted or specified with a value of "document" and all of whose `soapbind:body` elements specify the `use`="literal" attribute.

This profile places the following constraints message elements and on how conformant binding elements may use message element(s).

Use of `wsdl:message` elements with zero parts is permitted in Document styles to permit operations that can send or receive MESSAGEs with empty SOAP Bodies. Use of `wsdl:message` elements with zero parts is permitted in RPC styles to permit operations that have no (zero) parameters and/or return value. This case is explicitly permitted by the Basic Profile.
For Document-literal binding, the Basic Profile requires that at most one wsdl:part is marshaled in SOAP Body, which needs to be defined via the element form at the abstract level.

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 a single `wsdl:part`.*

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.*

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 document-literal 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).*

soapbind:fault, soapbind:header and soapbind:headerfault assume that style="document", since faults and headers do not contain parameters. This requirement is consistent with R2204, which requires that all parts in style="document" that are bound to soapbind:body be defined using the element attribute.

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.*

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 message specified in the portType is bound to soapbind:body or soapbind:header etc. as appropriate when using SOAP binding as defined in WSDL 1.1 section 3.

R2209 *A `wsdl:binding` in a DESCRIPTION SHOULD bind every `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`.*

The examples 4, 5 in section 3.1 of the WSDL 1.1 specification incorrectly show the use of 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>

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

CORRECT:

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

4.4 Port Types

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.4.

In WSDL 1.1, `portType` elements are used to group a set of abstract operations. This profile places the following constraints on the use of `portType` element(s):

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

R2301 *The order of the parts in a message in the DESCRIPTION MUST be the definitive order of the part elements on the wire for any part in an operation.*

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.*

Solicit-Response and Notification are not well defined by the WSDL 1.1 specification and the WSDL 1.1 specification defines bindings for the One-way and Request-response primitives only.

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

To promote interoperability operation name overloading in a portType is disallowed by the Basic Profile. Note however that a portType may name operations same as the ones in another portType.

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

WSDL representation of an RPC function can have 0 or 1 return value while the return value can be an instance of a complex type. The parameterOrder definition where provided, must be consistent with this as well.

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

If a part from the output message is omitted from the list of parts that is the value of the `parameterOrder` attribute, the single omitted 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.

WSDL 1.1 specification does not clearly state that both `type` and `element` attributes can not be specified to define a `part` in a `wsdl:message`. This loop-hole needs to be closed.

R2306 *A `wsdl:portType` in a DESCRIPTION MUST NOT refer to `wsdl:message`s that define `wsdl:part`s that specify both `type` and `element` attributes on the same `wsdl:part`.*

4.5 Bindings

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.5.

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

For interoperability the choice of bindings is limited to the well defined and most commonly used SOAP Binding. MIME, HTTP GET/POST bindings are not permitted by the Basic Profile.

R2401 *A `wsdl:binding` in a DESCRIPTION MUST use WSDL SOAP Binding as defined in section "3 SOAP Binding" of the WSDL 1.1 specification.*

4.6 Ports

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.6.

In WSDL 1.1 the port element specifies an address for binding on a portType, thus defining a communication end-point for the Web service. This profile places the following constraints on the use of the port element:

Editors' note: The Working Group has not closed any issues relating to Ports as of publication.

4.7 Services

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 2.7.

In WSDL 1.1 the service element is used to aggregate a set of related ports. This profile places the following constraints on the use of the service element:

Editors' note: The Working Group has not closed any issues relating to Services as of publication.

4.8 SOAP Binding

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- WSDL 1.1, Section 3.0.

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

SOAP 1.2 specification differs from the SOAP 1.1 specification in many respects. For interoperability the profile limits the SOAP binding to the SOAP 1.1 protocol.

R2700 *A `wsdl:binding` DESCRIPTION MUST use SOAP 1.1 protocol with SOAP Binding.*

Eliminate inconsistency between the WSDL 1.1 specification text and the WSDL 1.1 schema. The WSDL 1.1 specification shows it to be required but, the schema shows this attribute to be optional, where as the Basic profile sees this to be a required attribute.

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

For interoperability the transport protocol is limited to HTTP. To permit secure transfers at the HTTP level use of HTTP(S) is allowed.

R2702 *A `wsdl:binding` in a DESCRIPTION MUST specify the HTTP transport protocol with SOAP binding. Specifically, the `transport` attribute of is `soapbind:binding` child MUST have the value "http://schemas.xmlsoap.org/soap/http".*

Disallow mix and match of operation "style" in the same binding.

R2705 *A `wsdl:binding` in a DESCRIPTION MUST use the same value, either "rpc" or "document," for the `style` attribute for all of its `soapbind:operation`s.*

For interoperability the profile prohibits the use of different 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:header`, and `soapbind:headerfault` elements.*

Eliminate the ambiguity between the WDSL specification text and the Schema given in Appendix A4 about 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: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.*

Explicitly permit multiple bindings for the same portType, to alleviate any ambiguity in this area.

R2709 *A `wsdlportType` in a DESCRIPTION MAY have any number of `wsdl:binding`s that refer to it defined in the same or other WSDL documents.*

An endpoint that supports multiple operations has to identify unambiguously 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 Basic 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. The operations in a `wsdl:binding` in a DESCRIPTION MUST result in wire signatures that are different from one another.*

In the case of RPC/literal binding, the operation name is used as wrapper for the part accessors. However in the doc (literal) SOAP binding case since a wrapper with the operation name is not present, there might be an ambiguity as to which operation is invoked, if there are more than one operation in a port which is bound with doc/literal style. Hence the server needs to ensure it can distinguish the operation based on, on the wire SOAP message for the operation.

It is perceivable and it has also been demonstarted in the field that multiple Web services are hosted at the same network end-point by some implementations.

R2711 *A DESCRIPTION MAY have more than one `port` with the same value for the `location` attribute of the `soapbind:address` element.*

While it is not a conformance issue to have multiple `wsdl:port`s at the same network address, when the input messages destined for two different `wsdl:port`s are indistinguishable on the wire, the endpoint will not be able to determine the wsdl:port being invoked and this may result in implementation problems.

R2712 *When a Doc/literal binding is in use, the wire representation of a MESSAGE MUST be an instance of the global element declaration referenced by the corresponding `wsdl:message part`.*

Disambiguate the meaning of missing SOAPAction attribute specification in a WSDL description.

R2713 *If the value of the `soapAction` attribute on the `soapbind:operation` element is empty (as indicated by two quotes), the DESCRIPTION MUST be treated equivalent to the one that does not specify the `soapAction` attribute.*

HTTP being a request/response protocole, the profile needs to clarify what the expected behavior is in the case of a one-way message being exchanged.

Any SOAP content sent in reponse to a one-way operation would constitute a response message. Transmission of one-way operations MUST not result in processing level response or errors. It is therefore fobidden by the profile to send a SOAP envelop in reply to a one-way document. Hence a HTTP 500 "Internal Server Error" response that includes a SOAP message in the response containing a SOAP Fault element MUST NOT be returned.

There is no way at the HTTP level for the initiator of a request to know that the request was succesfully received until an HTTP response code is sent back. Based on the scemantics of the different response codes supported by the HTTP protocol, the profile specify that response codes 200 and 202 are the only response codes that the sender should interprete to signify that the message was succesfully delivered.

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

R2715 *INSTANCEs MUST NOT consider transmission of one-way operations complete until a HTTP response code of either "200 OK" or "202 accepted" is received by the HTTP client.*

R2727 *For one-way operations, INSTANCEs MUST NOT interpret the HTTP response code of 200 or 202 to mean the message is valid or that the receiver would process it.*

Despite the fact that the HTTP 1.1 specification assigns different meanings to responce codes 200 and 202, in the context of the basic profile there should be considered the same by the initiator of the request. This is due to the fact that some SOAP implementations have little control over the HTTP layer they depend on and might not be able to control the HTTP response code sent and when it is sent. As a result, the initiator of a one-way message should only interpret a 200 or 202 response code to mean that the message was succesfully received at the transport level but not that the SOAP processing layer and the application logic had a chance to validate the message or are commiting to processing it.

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 soapbind:body/@namespace attribute would override the element's namespace, which is prevented by the Basic Profile.

Conversely, in a rpc/literal SOAP binding, the serialized element child of the SOAP:Body is generated by wrapping the xsd Types referenced in the wsdl:part/@type attributes of the wsdl:message, into an element whose name is constructed from the value of the wsdl:part/@name attribute as the localname part, and whose assigned namespace is taken from the value of the soapbind:body/@namespace attribute. If the soapbind:body/@namespaceattribute were not specified, then the child of the SOAP:Body would not be namespace qualified.

Even for rpc/literal SOAP binding the soapbind:header, soapbind:headerfault and soapbind:fault assume that style="document", since faults and headers do not contain parameters. This requirement is consistent with R2716, which requires that all parts for style="document" not contain the namespace attribute.

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.*

The WSDL definition must be consistent at both abstract (portType) and concrete (binding) levels.

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

All known header faults for an operation should be idenfied in a WSDL description.

R2736 *A `wsdl:binding` in a DESCRIPTION SHOULD contain `soapbind:headerfault` elements describing each known header fault that might be generated.*

Eliminate inconsistency between WSDL specification text and the WSDL schema. WSDL 1.1 schema makes the specification of `soapbind:headerfault` element manadatory on `wsdl:input` and `wsdl:output` elements of an operation, where as the WSDL 1.1 specification marks them optional.

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

A Web service definition 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 indentified when the Web service was defined.

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

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

R2742 *A MESSAGE MAY contain fault detail elements in a `soap:Fault` element that are NOT described by a `wsdl:fault` element in the corresponding WSDL description.*

R2743 *A MESSAGE MAY contain fault detail elements in a `soap:Header` element that are NOT described by a `wsdl:headerfault` element in the corresponding WSDL description.*

The WSDL 1.1 schema is inconsistent with the WSDL 1.1 specification here and incorrectly names the attribute `parts` and gives a type of "NMTOKENS". The schema is incorrect since each `soapbind:header` element references a single part.

R2720 *A `wsdl:binding` in a DESCRIPTIONs MUST use the attribute named `part` with a Schema type of "NMTOKEN" on all contained `soapbind:header` and `soapbind:headerfault` elements. It MUST NOT use the attribute named `parts` on contained `soapbind:header` and `soapbind:headerfault` elements.*

For example,

CORRECT:

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

Eliminate inconsistency between WSDL1.1 specification text and the schema. The WSDL 1.1 schema 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.*

Eliminate inconsistency between WSDL 1.1 specification text and the schema.

WSDL 1.1 specification section 3.6 soap:Fault grammar, indicates that the `use` attribute of fault is required while in the schema the `use` attribute is defined as optional.

Also ensure the correct value is provided for the `use` attribute when present and identify the correct default value for the attribute when omitted.

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

R2723 *A `wsdl:binding` in a DESCRIPTION if 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

For interoperability, it is important to define how a conformant port should behave when a message that does not conform to its WSDL description is received.

R2724 *If an INSTANCE receives a message that is inconsistent with its WSDL description, it MUST return a `soap:Fault` with a faultcode of "VersionMismatch", or "MustUnderstand", or "Client".*

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

The description in section 3.5 of the WSDL 1.1 specification could be interpreted to mean the RPC response wrapper element must be named identical to the name of the `wsdl:operation`.

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

For rpc-literal SOAP messages, the WSDL 1.1 specification 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. Settling on one of the alternatives is crucial to achieving interoperability. The Basic Profile chose to place the part accessor elements in no namespace as it is simple, covers all cases and does not lead to logical inconsistency.

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

For rpc-literal SOAP messages, the WSDL 1.1 specification 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. The Basic Profile chose to qualify the child elements of the part accessor elements with the targetNamespace in which their types (elements and attributes) were defined.

R2737 *A MESSAGE described with an rpc-literal binding MUST namespace qualify the children of part accessor elements for parameters and return with the targetNamespace in which their types are defined.*

The WSDL1.1 spec in sect 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."

Yet, it doesn't 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 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:

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

CORRECT:

```
<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>
```

Editors' note: The examples above demonstrate the purpose of this requirement. The first segment shows a WSDL that defines some schema in the http://example.org/foo/

namespace in the wsdl:types section contained within a wsdl:definitions that has as its targetNamespace http://example.org/bar/. Thus, we have a type declared in one namespace and the containing element defined in another. The resulting SOAP message for BarOperation is shown in the segment that follows.

WSDL 1.1 specification is not clear if all `soapbind:headers` specified on `wsdl:input` or `wsd:output` of an `operation` in the SOAP binding section of a WSDL document must be included in the resultant SOAP messages transmitted. Basic Profile chose to make all the headers mandatory as there is no way in WSDL 1.1 to mark a header optional.

R2738 *A MESSAGE MUST include all `soapbind:header`s specified on `wsdl:input` or `wsd:output` of an `operation` `wsdl:binding` that describes it.*

Headers are SOAP's extensibility mechanism and headers that are not defined in the WSDL document may need to be included in the SOAP messages for various valid reasons described in SOAP.

R2739 *A MESSAGE MAY contain headers that are not described in the `wsdl:binding` that describes it. These headers MAY (or may not) have the mustUnderstand attribute set to '1'.*

Interoperability testing has demonstrated that requiring the value of the `SOAPAction` HTTP Header Field to be quoted increases interoperability of implementations. Even though HTTP allows for the value of HTTP Header Fields 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 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 MESSAGE MUST contain a `SOAPAction` HTTP Header Field with a quoted empty string, if in the corresponding WSDL description, the `soapAction` of `soapbind:operation` is either not present, or present with an empty string as its value.*

R2746 *A RECEIVER MAY respond with a Fault if the value of the `SOAPAction` HTTP Header Field is not quoted.*

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: ""
```

The wsdl:required attributed has been widely misunderstood and used by WSDL writers sometimes to incorrectly to indicate the optionality of soapbind:headers. 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 messages. For example, wsdl:required='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 messages generated from the WSDL description. It is meant to be interpreted as "in order to send a message 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, the extension elements may be ignored. The Basic Profile requires that all WSDL SOAP 1.1 extensions be understood and processed by the WSDL processors, irrespective of the presence or the value of the wsdl:required attribute on an extension element.

R2747 *A WSDL RECEIVER 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 WSDL RECEIVER 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 messages generated from the WSDL description.*

4.9 XML Schema

This portion of the profile modifies and refers to the following specifications (or sections thereof);

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

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

R2800 *DESCRIPTIONs MAY use any construct from XML Schema 1.0.*

R2801 *DESCRIPTIONs MUST use XML Schema 1.0 Recommendation, with the namespace URI "http://www.w3.org/2001/XMLSchema".*

5. Service Publication and Discovery

When publication or discovery of Web services is required, UDDI is the mechanism the Basic 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 Basic 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.

Editors' note: The Web services that constitute UDDI V2 are not fully conformant with the Basic Profile 1.0 because they do not accept messages 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 Basic Profile was developed. UDDI's designers are aware of UDDI V2's nonconformance and will take it into consideration in their future work.

This portion of the profile incorporates the following specifications by reference;

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

5.1 bindingTemplates

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- 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</accessPoint>
   <tModelInstanceDetails>
      ...
   </tModelInstanceDetails>
</bindingTemplate>
```

5.2 tModels

This portion of the profile modifies and refers to the following specifications (or sections thereof);

- 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 Basic 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.

It is not easy and in some cases it may be impossible to determine whether a given `uddi:tModel` represents a conformant Web service type by inspection alone because `uddi:tModel` elements describing conformant and non-conformant Web service types can look very similar. It needs to be easy for inquirers to determine whether a given `uddi:tModel` conforms and to discover conforming `uddi:tModel` elements.

Therefore the Basic 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.*

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.*

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".*

It would be ambiquous if the conformance claim a `uddi:tModel` made were not consistent with the claim made by the `wsdl:binding` it uses.

> R3004 *REGDATA of type `uddi:tModel` MUST be constructed so that the conformance claim it makes is consistent with the conformance claim made by the `wsdl:binding` to which it refers.*

The natural mechanism in UDDI for adding attributes to a `uddi:tModel` is to define and use a category system. The profile adopts this mechanism to add the ability for `uddi:tModel`s to assert conformance with WS-I profiles, and with the Basic Profile in particular.

> R3020 *REGDATA of type `uddi:tModel` claiming conformance with a WS-I profile MUST be categorized using the ws-i-org:conformsTo taxonomy.*

> R3030 *REGDATA of type `uddi:tModel` claiming conformance with a the Basic Profile MUST use the ws-i-org:conformsTo categorization value of "http://www.ws-i.org/profiles/base/1.0".*

For example,

CORRECT:

```
<tModel tModelKey="...">
   <name>BarSOAPService</name>
   <description xml:lang="EN">Barâ€™s SOAP Service</description>
```

```
<overviewDoc>...</overviewDoc>
<categoryBag>
   <keyedReference
      tModelKey="uddi:..."
      keyName="ws-I_conformance:BasicProfile1.0"
      keyValue="http://www.ws-i.org/profiles/basic/1.0" />
</categoryBag>
```

Editors' note: The value of the categorization in R3030 is intended to match the value used in the resolution of issue w27 concerning how to mark WSDL elements that conform to the profile. If the value representing the Basic Profile 1.0 changes in the final resolution, the value used here should be changed to match.

Since `wsdl:service` elements are not necessarily mapped to a single `uddi:businessService` and are also not subject to conformance claims, it would be unclear what it meant if a `uddi:businessEntity` or a `uddi:businessService` element were to claim conformance with the Basic Profile. Also, `uddi:bindingTemplate` elements can't be categorized because the UDDI V2 XML Schema does not provide a `uddi:categoryBag` for them. Hence, the conformance claim made by `wsdl:port` elements can't be documented in the corresponding `uddi:bindingTemplate`.

R3005 *REGDATA other than `uddi:tModel` elements representing conformant Web service types MUST NOT be categorized using the ws-i-org:conformsTo taxonomy and a categorization of "http://www.ws-i.org/profiles/base/1.0".*

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 Basic 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 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 which 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 portion of the profile incorporates the following specifications by reference;

- 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.

6.1 The 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 *INSTANCEs MAY require the use of HTTPS with mutual authentication.*

6.2 Certificate Authority

Successful use of basic HTTPS requires the consumer to agree that the instance's certificate was issued by an acceptable authority. Successful use of mutual authentication additionally requires the instance to agree that the consumer's certificate was issued by an acceptable authority. The choice of which certificate authorities are acceptable is an important consideration in the effectiveness of using HTTPS, but is a policy decision that is beyond the scope of the profile.

R5100 *If an INSTANCE requires use of basic HTTPS, it MUST choose a certificate authority for its certificate that is acceptable to the consumer.*c

R5110 *If an INSTANCE requires the use of HTTPS with mutual authentication, it MUST find the choice of certificate authority for the consumer's certificate acceptable.*c

6.3 Permitted HTTPS Encryption Algorithms

Successful use of HTTPS requires the consumer and the instance to agree on a mutually acceptable encryption algorithm. The choice of which encryption algorithms are acceptable is an important consideration in the effectiveness of using HTTPS, but is a policy decision that is beyond the scope of the profile.

R5200 *If an INSTANCE requires the use of HTTPS, it MUST choose an encryption algorithm that is acceptable to the consumer.*c