# Web Services Description Language (WSDL) 1.0

## 25 September 2000

Authors (alphabetically):

Erik Christensen, Microsoft
Francisco Curbera, IBM
Greg Meredith, Microsoft
Sanjiva Weerawarana, IBM

---

## Abstract

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

This version of the WSDL language is a first step which does not include a framework for describing the composition and orchestration of endpoints. A complete framework for describing such contracts will include means for composing services and means for expressing the behavior of services, i.e. the sequencing rules for sending and receiving messages. Composition of services must be type safe but also allow for reference passing with service references being exchanged and bound at runtime. The latter being key for negotiating contracts at runtime and capturing the behavior of referral and brokering services.

The authors of the WSDL specification intend to publish revised versions of WSDL and/or additional documents in a timely fashion which will include a (1) framework for composing services and a (2) framework for describing the behavior of services.

## Status

This draft represents the current thinking with regard to descriptions of services within Ariba, IBM and Microsoft.  It consolidates concepts found in NASSL, SCL, and SDL (earlier proposals in this space).

## Table of Contents

1 Introduction.
1.1 WSDL Document Example
1.2 Notational Conventions
2 Service Definition

2.1 Document Structure
2.1.1 Document Naming and Linking
2.1.2 Athoring Style
2.1.3 Language Extensibility and Binding
2.1.4 Documentation
2.2 Types
2.3 Messages
2.3.1 Message Parts
2.3.2 Abstract vs. Concrete Messages
2.4 Port Types
2.4.1 One-way Operation
2.4.2 Request-response Operation.
2.4.3 Solicit-response Operation
2.4.4 Notification Operation
2.4.5 Names of Elements within an Operation
2.4.6 Parameter Order within an Operation
2.5 Bindings
2.6 Ports
2.7 Services
3 SOAP Binding
3.1 SOAP Examples
3.2 How the SOAP Binding Extends WSDL
3.3 soap:binding
3.4 soap:operation
3.5 soap:body
3.6 soap:fault
3.7 soap:header
3.8 soap:address
4 HTTP GET & POST Binding
4.1 HTTP GET/POST Examples
4.2 How the HTTP GET/POST Binding Extends WSDL
4.3 http:address
4.4 http:binding
4.5 http:operation
4.6 http:urlEncoded
4.7 http:urlReplacement
5 MIME Binding
5.1 MIME Binding example
5.2 How the MIME Binding extends WSDL
5.3 mime:content
5.4 mime:multipartRelated
5.5 soap:body
5.6 mime:mimeXml
6 References
A 1 Notes on URIs
A 1.1 XML namespaces & schema locations
A 1.2 Relative URIs
A 1.3 Generating URIs
A 2 Wire format for WSDL examples
A 2.1 Example 1
A 3 Location of Extensibility Elements
A 4 Schemas
A 4.1 WSDL Schema
A 4.2 SOAP Binding Schema
A 4.3 HTTP Binding Schema
A 4.4 MIME Binding Schema

# 1. Introduction

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **Types**– a container for data type definitions using some type system (such as XSD).
- **Message**– an abstract, typed definition of the data being communicated.
- **Operation**– an abstract description of an action supported by the service.
- **Port Type**–an abstract set of operations supported by one or more endpoints.
- **Binding**– a concrete protocol and data format specification for a particular port type.
- **Port**– a single endpoint defined as a combination of a binding and a network address.
- **Service**– a collection of related endpoints.

These elements are described in detail in Section 2. It is important to observe that WSDL does not introduce a new type definition language. WSDL recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification (XSD) [11] as its canonical type system. However, since it is unreasonable to expect a single type system grammar to be used to describe all message formats present and future, WSDL allows using other type definition languages via extensibility.

In addition, WSDL defines a common **binding** mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions.

In addition to the core service definition framework, this specification introduces specific **binding extensions** for the following protocols and message formats:

- SOAP 1.1 (see Section 3)
- HTTP GET / POST (see Section 4)
- MIME (see Section 5)

Although defined within this document, the above language extensions are layered on top of the core service definition framework. Nothing precludes the use of other binding extensions with WSDL.

## 1.2 WSDL Document Example

The following example shows the WSDL definition of a simple service providing stock quotes. The service supports a single operation called GetLastTradePrice, which is deployed using the SOAP 1.1 protocol over HTTP. The request takes a ticker symbol of type string, and returns the price as a float. A detailed description of the elements used in this definition can be found in Section 2 (core language) and Section 3 (SOAP binding).

**Example 1 SOAP 1.1 Request/Response via HTTP**

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/1999/XMLSchema">
           <element name="TradePriceRequest">
```

```
                <complexType>
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
             </element>
             <element name="TradePrice">
                <complexType>
                    <all>
                        <element name="price" type="float"/>
                    </all>
                </complexType>
             </element>
         </schema>
    </types>

    <message name="GetLastTradePriceInput">
         <part name="body" element="xsd1:TradePrice"/>
     </message>

     <message name="GetLastTradePriceOutput">
         <part name="body" element="xsd1:TradePriceResult"/>
     </message>

     <portType name="StockQuotePortType">
         <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
         </operation>
     </portType>

     <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
         <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
         <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
            <input>
                <soap:body use="literal" namespace="http://example.com/stockquote.xsd"
                           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="literal" namespace="http://example.com/stockquote.xsd"
                           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
         </operation>
     </binding>

     <service name="StockQuoteService">
         <documentation>My first service</documentation>
         <port name="StockQuotePort" binding="tns:StockQuoteBinding">
            <soap:address location="http://example.com/stockquote"/>
         </port>
     </service>

</definitions>
```

## 1.2 Notational Conventions

1. The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [2].

2. The following namespace prefixes are used throughout this document:

| prefix | namespace URI | definition |
|---|---|---|
| wsdl | http://schemas.xmlsoap.org/wsdl/ | WSDL namespace for WSDL framework. |
| http | http://schemas.xmlsoap.org/wsdl/http/ | WSDL namespace for HTTP GET & POST binding. |
| mime | http://schemas.xmlsoap.org/wsdl/mime/ | WSDL namespace for MIME binding. |

| | | |
|---|---|---|
| xsi | http://www.w3.org/1999/XMLSchema-instance | Instance namespace as defined by XSD [11]. |
| xsd | http://www.w3.org/1999/XMLSchema | Schema namespace as defined by XSD [11]. |
| tns | (various) | The “this namespace” (tns) prefix is used as a convention to refer to the current document. |
| (other) | (various) | All other namespace prefixes are samples only. In particular, URIs starting with “http://example.com” represent some application-dependent or context-dependent URI [4]. |

3. This specification uses an **informal syntax** to describe the XML grammar of a WSDL document:

- The syntax appears as an XML instance, but the values indicate the data types instead of values.
- Characters are appended to elements and attributes as follows: "?" (0 or 1), "*" (0 or more), "+" (1 or more).
- Elements names ending in "…" (such as <element…/> or <element…>) indicate that elements/attributes irrelevant to the context are being omitted.
- Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.
- <-- extensibility element --> is a placeholder for elements from some "other" namespace (like ##other in XSD).
- The XML namespace prefixes (defined above) are used to indicate the namespace of the element being defined.
- Examples starting with <?xml contain enough information to conform to this specification; others examples are fragments and require additional information to be specified in order to conform.

XSD schemas are provided as a formal definition of WSDL grammar:

# 2. Service Definition

This section describes the core elements of the WSDL language. Binding extensions for SOAP, HTTP and MIME are included in Sections 3, 4 and 5.

## 2.1 WSDL Document Structure

A WSDL document is simply a **set of definitions**. There is a **definitions** element at the root, and definitions inside. The grammar is as follows:

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri"?>

    <import namespace="uri" location="uri"/>*

    <wsdl:documentation…/> ?

    <wsdl:types> ?
        <wsdl:documentation…/>?
        <xsd:schema…/>*
        <-- extensibility element --> *
    </wsdl:types>

    <wsdl:message name="nmtoken> *
        <wsdl:documentation…/>?
        <part name="nmtoken" element="qname"? type="qname"?/> *
    </wsdl:message>

    <wsdl:portType name="nmtoken">*
        <wsdl:documentation…/>?
        <wsdl:operation name="nmtoken">*
           <wsdl:documentation…/> ?
           <wsdl:input name="nmtoken"? message="qname">?
               <wsdl:documentation…/> ?
```

```
            </wsdl:input>
            <wsdl:output name="nmtoken"? message="qname">?
                <wsdl:documentation…/> ?
            </wsdl:output>
            <wsdl:fault name="nmtoken" message="qname"> *
                <wsdl:documentation…/> ?
            </wsdl:fault>
         </wsdl:operation>
     </wsdl:portType>

     <wsdl:binding name="nmtoken" type="qname">*
         <wsdl:documentation…/>?
         <-- extensibility element --> *
         <wsdl:operation name="nmtoken">*
            <wsdl:documentation…/> ?
            <-- extensibility element --> *
            <wsdl:input name="nmtoken"?> ?
                <wsdl:documentation…/> ?
                <-- extensibility element -->
            </wsdl:input>
            <wsdl:output name="nmtoken"?> ?
                <wsdl:documentation…/> ?
                <-- extensibility element --> *
            </wsdl:output>
            <wsdl:fault name="nmtoken"> *
                <wsdl:documentation…/> ?
                <-- extensibility element --> *
            </wsdl:fault>
         </wsdl:operation>
     </wsdl:binding>

     <wsdl:service name="nmtoken"> *
         <wsdl:documentation…/>?
         <wsdl:port name="nmtoken" binding="qname"> *
            <wsdl:documentation…/> ?
            <-- extensibility element -->
         </wsdl:port>
         <-- extensibility element -->
     </wsdl:service>

     <-- extensibility element --> *

</wsdl:definitions>
```

Service are defined using five major elements:

- **types**, which provides data type definitions used to describe the messages exchanged.
- **message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **portType**, which is a set of abstract operations. Each operation refers to an input message and output messages.
- **binding**, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- **port**, which specifies an address for a binding, thus defining a single communication endpoint.
- **service**, which is used to aggregate a set of related ports.

These elements will be described in detail in Sections 2.2 to 2.7. In the rest of this section we describe the rules introduced by WSDL for naming documents, referencing document definitions, using language extensions and adding contextual documentation.

### 2.1.1 Document Naming and Linking

WSDL documents can be assigned an optional **name** attribute of type NCNAME that serves as a lightweight form of documentation. Optionally, a **targetNamespace** attribute of type URI may be specified. The URI MAY NOT be a relative URI.

WSDL allows associating a **namespace** with a document **location** using an **import** statement:

```
<definitions…>
    <import namespace="uri" location="uri"/> *
</definitions>
```

The resolution of QNames in WSDL is similar to the resolution of QNames described by the XML Schemas specification [11].

A reference to a WSDL definition is made using a QName. The following types of definitions contained in a WSDL document may be referenced:

- WSDL definitions: service, port, message, bindings, and portType
- Other definitions: if additional definitions are added via extensibility, they SHOULD use QName linking.

Each WSDL definition type listed above has its own **name scope** (i.e. port names and message names never conflict). ames within a name scope MUST be unique within the WSDL document.

## 2.1.2 Authoring Style

The use of the **import** element allows the separation of the different elements of a service definition into independent documents, which can then be imported as needed. This technique helps writing clearer service definitions, by separating the definitions according to their level of abstraction. It also maximizes the ability to reuse service definitions of all kinds. As a result, WSDL documents structured in this way are easier to use and maintain. Example 2 below shows how to use this authoring style to define the service presented in Example 1. Here we separate the definitions in three documents: data type definitions, abstract definitions, and specific service bindings. The use of this mechanism is of course not limited to the definitions explicitly presented in the example, which uses only language elements defined in this specification. Other types of definitions based on additional language extensions can be encoded and reused in a similar fashion.

**Example 2. Alternative authoring style for the service in Example 1.**

http://example.com/stockquote/stockquote.xsd

```
<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
       xmlns="http://www.w3.org/1999/XMLSchema">
    <element name="GetLastTradePrice">
        <complexType>
           <all>
               <element name="tickerSymbol" type="string"/>
           </all>
        </complexType>
   </element>
   <element name="GetLastTradePriceResult">
        <complexType>
           <all>
               <element name="result" type="float"/>
           </all>
        </complexType>
   </element>
</schema>
```

http://example.com/stockquote/stockquote.wsdl

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
          xmlns:tns="http://example.com/stockquote/definitions"
          xmlns:xsd1="http://example.com/stockquote/schemas"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

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

   <message name="GetLastTradePriceRequest">
        <part name="body"element="xsd1:GetLastTradePrice"/>
   </message>

   <message name="GetLastTradePriceResponse">
        <part name="body"element="xsd1:GetLastTradePriceResult"/>
   </message>

   <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceRequest"/>
           <output message="tns:GetLastTradePriceResponse"/>
       </operation>
   </portType>
```

```
</definitions>
```

http://example.com/stockquote/stockquoteservice.wsdl

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/service"
          xmlns:tns="http://example.com/stockquote/service"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

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

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
         <soap:binding style="document"/>
         <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://my.org/GetLastTradePrice"/>
         </operation>>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://my.org/stockquote"/>
        </port>
   </service>
</definitions>
```

### 2.1.3 Language Extensibility and Binding

In WSDL the term binding refers to the process associating protocol or data format information with an abstract entity like a message, operation, or portType. WSDL allows elements representing a specific technology (referred to here as **extensibility elements**) under various elements defined by WSDL. These points of extensibility are typically used to specify binding information for a particular protocol or message format, but are not limited to such use. Extensibility elements MUST use an XML namespace different from that of WSDL. The specific locations in the document where extensibility elements can appear are described in detail in Appendix A3.

Extensibility elements are commonly used to specify some technology specific binding. To distinguish whether the semantic of the technology specific binding is required for communication or optional, extensibility elements MAY place a **wsdl:required** attribute of type boolean on the element. The default value for required is false. The required attribute is defined in the namespace "http://schemas.xmlsoap.org/wsdl/".

See Sections 3, 4, and 5 for examples of extensibility elements.

### 2.1.4 Documentation

WSDL uses the optional **wsdl:document** element as a container for human readable documentation. The content of the element is arbitrary text and elements ("mixed" in XSD). The documentation element is allowed inside any WSDL language element.

## 2.2 Types

The **types** element encloses data type definitions that are relevant for the exchanged messages. For maximum interoperability and platform neutrality, WSDL prefers the use of XSD as the canonical type system, and treats it as the intrinsic type system.

```
<definitions…>
    <types>
        <xsd:schema…/>*
    </types>
</definitions>
```

The XSD type system can be used to define the types in a message regardless of whether or not the resulting wire format is actually XML, or whether the resulting XSD schema validates the particular wire format. This is especially interesting if there will be multiple bindings for the same message, or if there is only one binding but that binding type does not already have a type system in widespread use. In these cases, the recommended approach for encoding abstract types using XSD is as follows:

- Use element form (not attribute).
- Don't include attributes or elements that are peculiar to the wire encoding (e.g. have nothing to do with the abstract content

of the message). Some examples are soap:root, soap:encodingStyle, xmi:id, xmi:name.

- Use the Soap:Array type for modeling array types (regardless of whether the resulting form actually uses the encoding specified in Section 5 of the SOAP v1.1 document). Use the name ArrayOfXXX for array types (where XXX is the type of the items in the array).

However, since it is unreasonable to expect a single type system grammar can be used to describe all abstract types present and future, WSDL allows type systems to be added via extensibility elements. An extensibility element may appear under the **types** element to identify the type definition system being used and to provide an XML container element for the type definitions. The role of this element can be compared to that of the **schema** element of the XML Schema language.

```
<definitions…>
    <types>
        <-- type-system extensibility element --> *
    </types>
</definitions>
```

## 2.3 Messages

Messages consist of one or more logical **parts**. Each part is associated with a type from some type system using a message-typing attribute. The set of message-typing attributes is extensible. WSDL defines several such message-typing attributes for use with XSD:

- **element**. Refers to an XSD element using a QName.
- **type**. Refers to an XSD simpleType or complexType using a QName.

Other message-typing attributes may be defined as long as they use a namespace different from that of WSDL. Binding extensibility elements may also use message-typing attributes.

The syntax for defining a message is as follows. The message-typing attributes (which may vary depending on the type system used) are shown in **bold**.

```
<definitions…>
    <message name="nmtoken"> *
        <part name="nmtoken"? element="qname" type="qname"?/> *
    </message>
</definitions>
```

The message **name** attribute provides a unique name among all messages defined within the enclosing WSDL document.

The part **name** attribute provides a unique name among all the parts of the enclosing message.

### 2.3.1 Message Parts

Parts are a flexible mechanism for describing the logical abstract content of a message. A binding may reference the ame of a part in order to specify binding-specific information about the part. For example, if defining a message for use with RPC, a part MAY represent a parameter in the message. However, the bindings must be inspected in order to determine the actual meaning of the part.

Multiple part elements are used if the message has multiple logical units. For example, the following message consists of a Purchase Order and a Customer.

```
<definitions…>
    <types>
        <schema…>
           <element name="PO" type="tns:POType"/>
           <complexType name="POType">
               <element name="id" type="string/>
               <element name="name" type="string"/>
               <element name="items">
                   <complexType>
                       <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
                   </complexType>
               </element>
           </complexType>

           <complexType name="Item">
               <element name="quantity" type="int"/>
               <element name="product" type="string"/>
           </complexType>
           <element name="Customer" type="tns:CustomerType"/>
```

```
            <complexType name="CustomerType">
                <element name="name" type="string"/>
            </complexType>
        </schema>
    </types>

    <message name="PO">
        <part name="po" element="tns:PO"/>
        <part name="customer" element="tns:Customer"/>
    </message>
</definitions>
```

However, if the message contents are sufficiently complex, then an alternative syntax may be used to specify the composite structure of the message using the type system directly. In the following example, the body is either a purchase order, or a set of customers.

```
<definitions…>
    <types>
        <schema…>
            <complexType name="POType">
                <element name="id" type="string/>
                <element name="name" type="string"/>
                <element name="items">
                    <complexType>
                        <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
                    </complexType>
                </element>
            </complexType>

            <complexType name="Item">
                <element name="quantity" type="int"/>
                <element name="product" type="string"/>
            </complexType>
            <complexType name="CustomerType">
                <element name="name" type="string"/>
            </complexType>

            <complexType name="Composite">
                <choice>
                    <element name="PO" minOccurs="1" maxOccurs="1" type="tns:POType"/>
                    <element name="Customer" minOccurs="0" maxOccurs="unbounded" type="tns:CustomerType"/>
                </choice>
            </complexType>
        </schema>
    </types>

    <message name="PO">
        <part name="composite" type="tns:Composite"/>
    </message>
</definitions>
```

### 2.3.2 Abstract vs. Concrete Messages

Message definitions are always considered to be an abstract definition of the message content. A message binding describes how the abstract content is mapped into a concrete format. However, in some cases, the abstract definition may match the concrete representation very closely or exactly for one or more bindings, so those binding(s) will supply little or no mapping information. However, another binding of the same message definition may require extensive mapping information. For this reason, it is not until the binding is inspected that one can determine "how abstract" the message really is.

## 2.4 Port Types

A port type is a named set of abstract operations and the abstract messages involved.

```
<wsdl:definitions…>
    <wsdl:portType name="nmtoken">
        <wsdl:operation name="nmtoken"…/> *
    </wsdl:portType>
</wsdl:definitions>
```

The port type **name** attribute provides a unique name among all port types defined within in the enclosing WSDL document.

An operation is named via the **name** attribute.

WSDL has four transmission primitives that an endpoint can support:

- **One-way**. The endpoint receives a message.
- **Request-response**. The endpoint receives a message, and sends a correlated message.
- **Solicit-response**. The endpoint sends a message, and receives a correlated message.
- **Notification**. The endpoint sends a message.

WSDL refers to these primitives as **operations**. Although request/response or solicit/response can be modeled abstractly using two one-way messages, it is useful to model these as primitive operation types because:

- They are very common.
- The sequence can be correlated without having to introduce more complex flow information.
- Some endpoints can only receive messages if they are the result of a synchronous request response.
- A simple flow can algorithmically be derived from these primitives at the point when flow definition is desired.

Although request/response or solicit/response are logically correlated in the WSDL document, a given binding describes the concrete correlation information. For example, the request and response messages may be exchanged as part of one or two actual etwork communications.

Operations refer to the messages involved using the **message** attribute of type QName. This attribute follows the rules defined by WSDL for linking (see section 2.1.2).

## 2.4.1 One-way Operation

The grammar for a one-way operation is:

```
<wsdl:definitions…> <wsdl:portType…> *
        <wsdl:operation name="nmtoken">
           <wsdl:input name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

The **input** element specifies the abstract message format for the one-way operation.

## 2.4.2 Request-response Operation

The grammar for a request-response operation is:

```
<wsdl:definitions…>
    <wsdl:portType…> *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:input name="nmtoken"? message="qname"/>
           <wsdl:output name="nmtoken"? message="qname"/>
           <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

The input and output elements specify the abstract message format for the request and response, respectively. The optional fault elements specify the abstract message format for any error messages that may be output as the result of the operation (beyond those specific to the protocol).

Note that a request-response operation is an abstract notion; a particular binding must be consulted to determine how the messages are actually sent: within a single communication (such as a HTTP request/response), or as two independent communications (such as two HTTP requests).

## 2.4.3 Solicit-response Operation

The grammar for a solicit-response operation is:

```
<wsdl:definitions…>
    <wsdl:portType…> *
        <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
           <wsdl:output name="nmtoken"? message="qname"/>
```

```
            <wsdl:input name="nmtoken"? message="qname"/>
            <wsdl:fault name="nmtoken" message="qname"/>*
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

The output and input elements specify the abstract message format for the solicited request and response, respectively. The optional fault elements specify the abstract message format for any error messages that may be output as the result of the operation (beyond those specific to the protocol).

Note that a request-response operation is an abstract notion; a particular binding must be consulted to determine how the messages are actually sent: within a single communication (such as a HTTP request/response), or as two independent communications (such as two HTTP requests).

### 2.4.4 Notification Operation

The grammar for a one-way operation is:

```
<wsdl:definitions…>
    <wsdl:portType…> *
        <wsdl:operation name="nmtoken">
           <wsdl:output name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

The **output** element specifies the abstract message format for the notification operation.

### 2.4.5 Names of Elements within an Operation

The **name** attribute of the input and output elements provides a unique name among all input and output elements within the enclosing port type.

In order to avoid having to name each input and output element within an operation, WSDL provides some default values based on the operation name. If the name attribute is not specified on a one-way or notification message, it defaults to the name of the operation. If the name attribute is not specified on the input or output messages of a request-response or solicit-response operation, the name defaults to the name of the operation with "Request"/"Solicit" or "Response" appended, respectively.

Each fault element must be named to allow a binding to specify the concrete format of the fault message. The name of the fault element is unique within the set of faults defined for the operation.

### 2.4.6 Parameter Order within an Operation

Operations do not specify whether they are to be used with RPC-like bindings or not. However, when using an operation with an RPC-binding, it is useful to be able to capture the original RPC function signature. For this reason, a request-response or solicit-response operation MAY specify a list of parameter names via the **parameterOrder** attribute (of type nmtokens). The value of the attribute is a list of message part names separated by a single space. The parts named MUST follow the following rules:

- The order the parts are named to reflect the order of the parameters in the RPC signature
- The **return** value part is not present in the list
- If a part name appears in both the input and output message, it is an **in/out** parameter
- If a part name appears in only the input message, it is an **in** parameter
- If a part name appears in only the output message, it is an **out** parameter

Note that this information serves as a "hint" and may safely be ignored by those not concerned with RPC signatures. Also, it is not required to be present, even if the operation is to be used with an RPC-like binding.

## 2.5 Bindings

A binding defines message format and protocol details for operations and messages defined by a particular portType. There may be any number of bindings for a given portType. The grammar for a binding is as follows:

```
<wsdl:definitions…>
    <wsdl:binding name="nmtoken" type="qname"> *
        <-- extensibility element (1) --> *
        <wsdl:operation name="nmtoken"> *
           <-- extensibility element (2) --> *
           <wsdl:input name="nmtoken"?> ?
               <-- extensibility element (3) -->
           </wsdl:input>
           <wsdl:output name="nmtoken"?> ?
               <-- extensibility element (4) --> *
           </wsdl:output>
           <wsdl:fault name="nmtoken"> *
               <-- extensibility element (5) --> *
           </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
</wsdl:definitions>
```

The **name** attribute provides a unique name among all bindings defined within in the enclosing WSDL document.

A binding references the portType that it binds using the **type** attribute. This QName value follows the linking rules defined by WSDL (see section 2.1.2).

The binding's operation, input, output, and fault elements are correlated with the corresponding portType elements using the **name** attribute of each element, which behave exactly as within portType (see section 2.4.5).

Binding extensibility elements are used to specify the concrete grammar for the input (3), output (4), and fault messages (5). Per-operation binding information (2) as well as per-binding information (1) may also be specified.

A binding MUST specify exactly one protocol.

A binding MAY NOT specify address information.

## 2.6 Ports

A port defines an individual endpoint by specifying a single address for a binding.

```
<wsdl:definitions…>
    <wsdl:service…> *
        <wsdl:port name="nmtoken" binding="qname"> *
           <-- extensibility element (1) -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

The **name** attribute provides a unique name among all ports defined within in the enclosing WSDL document.

The **binding** attribute (of type QName) refers to the binding using the linking rules defined by WSDL (see Section 2.1.2).

Binding extensibility elements (1) are used to specify the address information for the port.

A port MAY NOT specify more than one address.

A port MAY NOT specify any binding information other than address information.

## 2.7 Services

A service groups a set of related ports together:

```
<wsdl:definitions…>
    <wsdl:service name="nmtoken"> *
        <wsdl:port…/>*
    </wsdl:service>
</wsdl:definitions>
```

The **name** attribute provides a unique name among all services defined within in the enclosing WSDL document.

Ports within a service have the following relationship:

- None of the ports communicate with each other (e.g. the output of one port is not the input of another).
- If a service has several ports that share a port type, but employ different bindings or addresses, the ports are alternatives.

Each port provides semantically equivalent behavior (within the transport and message format limitations imposed by each binding). This allows a consumer of a WSDL document to choose particular port(s) to communicate with based on some criteria (protocol, distance, etc.).

- By examining it's ports, we can determine a service's port types. This allows a consumer of a WSDL document to determine if it wishes to communicate to a particular service based whether or not it supports several port types. This is useful if there is some implied relationship between the operations of the port types, and that the entire set of port types must be present in order to accomplish a particular task.

# 3. SOAP Binding

WSDL includes a binding for SOAP 1.1 endpoints, which supports the specification of the following protocol specific information:

- An indication that a binding is bound to the SOAP 1.1 protocol
- A way of specifying an address for a SOAP endpoint.
- The URI for the SOAPAction HTTP header for the HTTP binding of SOAP
- A list of definitions for Headers that are transmitted as part of the SOAP Envelope
- A way of specifying SOAP roots in XSD

This binding grammar it is not an exhaustive specification since the set of SOAP bindings is evolving. Nothing precludes additional SOAP bindings to be derived from portions of this grammar. For example:

- SOAP bindings that do not employ a URI addressing scheme may substitute another addressing scheme by replacing the soap:address element defined in section 3.8.
- SOAP bindings that do not require a SOAPAction omit the soapAction attribute defined in section 3.4.

## 3.1 SOAP Examples

In the following example, a SubscribeToQuotes SOAP 1.1 one-way message is sent to a StockQuote service via a SMTP binding. The request takes a ticker symbol of type string, and includes a header defining the subscription URI.

**Example 3. SOAP binding of one-way operation over SMTP using a SOAP Header**

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <message name="SubscribeToQuotes">
        <part name="body" element="xsd1:SubscribeToQuotes"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://example.com/smtp"/>
        <operation name="SubscribeToQuotes">
           <input message="tns:SubscribeToQuotes">
               <soap:header element="xsd1:SubscriptionHeader"/>
           </input>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <port name="StockQuotePort" binding="tns:StockQuoteSoap">
           <soap:address location="mailto://subscribe@example.com"/>
        </port>
    </service>
    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
               xmlns="http://www.w3.org/1999/XMLSchema">
           <element name="SubscribeToQuotes">
               <complexType>
                   <all>
                       <element name="tickerSymbol" type="string"/>
                   </all>
               </complexType>
           </element>
           <element name="SubscriptionHeader" type="uriReference"/>
        </schema>
    </types>
</definitions>
```

This example describes that a GetLastTradePrice SOAP 1.1 request may be sent to a StockQuote service via the SOAP 1.1 HTTP binding. The request takes a ticker symbol of type string, a time of type timeInstant, and returns the price as a float in the SOAP response.

**Example 4. SOAP binding of request-response RPC operation over HTTP**

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <message name="GetLastTradePriceRequest">
        <part name="tickerSymbol" element="xsd:string"/>
        <part name="time" element="xsd:timeInstant"/>
    </message>

    <message name="GetLastTradePriceResponse">
        <part name="result" type="xsd:float"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceRequest"/>
           <output message="tns:GetLastTradePriceResponse"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="encoded" namespace="http://example.com/stockquote"
                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </input>
           <output>
               <soap:body use="encoded" namespace="http://example.com/stockquote"
                          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
           </output>
        </operation>>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

## 3.2 How the SOAP Binding Extends WSDL

The SOAP Binding extends WSDL with the following extension elements:

```
<definitions…>
```

```
    <binding…>
        <soap:binding style="rpc|document" transport="uri">
        <operation…>
           <soap:operation soapAction="uri"? style="rpc|document"?>?
           <input>
               <soap:body parts="nmtokens"? use="literal|encoded"
                          encodingStyle="uri-list"? namespace="uri"?>
               <soap:header element="qname" fault="qname"?>*
           </input>
           <output>
               <soap:body parts="nmtokens"? use="literal|encoded"
                          encodingStyle="uri-list"? namespace="uri"?>
               <soap:header element="qname" fault="qname"?>*
           </output>
           <fault>*
               <soap:fault name="nmtoken" use="literal|encoded"
                           encodingStyle="uri-list"? namespace="uri"?>
            </fault>
        </operation>
    </binding>

    <port…>
        <soap:address location="uri"/>
    </port>
</definitions>
```

Each extension element of the SOAP binding is covered in subsequent sections.

## 3.3 soap:binding

The purpose of the SOAP binding element is to signify that the binding is bound to the SOAP protocol format: Envelope, Header and Body. This element makes no claims as to the encoding or format of the message (e.g. that it necessarily follows section 5 of the SOAP 1.1 specification).

The soap:binding element MUST be present when using the SOAP binding.

```
<definitions…>
    <binding…>
        <soap:binding transport="uri"? style="rpc|document"?>
    </binding>
</definitions>
```

The value of the **style** attribute is the default for the style attribute for each contained operation. If the style attribute is omitted, it is assumed to be "document". See section 3.4 for more information on the semantics of style.

The value of the required **transport** attribute indicates which transport of SOAP this binding corresponds to. The URI value **http://schemas.xmlsoap.org/soap/http** corresponds to the HTTP binding in the SOAP specification. Other URIs may be used here to indicate other transports (such as SMTP, FTP, etc.).

## 3.4 soap:operation

The soap:operation element provides information for the operation as a whole.

```
<definitions…>
    <binding…>
        <operation…>
           <soap:operation soapAction="uri"? style="rpc|document"?>
        </operation>
    </binding>
</definitions>
```

The **style** attribute indicates whether the operation is RPC-oriented (messages containing parameters and return values) or document-oriented (message containing document(s)). This information may be used to select an appropriate programming model. If the attribute is not specified, it defaults to the value specified in the soap:binding element. If the soap:binding element does not specify a style, it is assumed to be "document".

The **soapAction** attribute specifies the value of the SOAPAction header for this operation. This URI value should be used directly as the value for the SOAPAction header; no attempt should be made to make a relative URI value absolute when making the request. For the HTTP protocol binding of SOAP, this value required (it has no default value). For other SOAP protocol bindings, it MAY NOT be specified, and the soap:operation element MAY be omitted.

## 3.5 soap:body

The soap:body element specifies how the message parts appear inside the SOAP Body element.

The parts of a message may either be abstract type definitions, or concrete schema definitions. If abstract definitions, the types are serialized according to some set of rules defined by an encoding style. Each encoding style is identified using a list of URIs, as in the SOAP specification. Since some encoding styles such as the SOAP Encoding (http://schemas.xmlsoap.org/soap/encoding/) allow variation in the message format for a given set of abstract types, it is up to the reader of the message to understand all the format variations: "reader makes right". To avoid having to support all variations, a message may be defined concretely and then indicate it's original encoding style (if any) as a hint. In this case, the writer of the message must conform exactly to the specified schema: "writer makes right".

The soap:body element is used to define both RPC-oriented and document-oriented messages. If the enclosing operation's style is "rpc", each part is a parameter or return value and appears inside a wrapper element within the Body (see Section 7.1 of the SOAP specification). If the style is "document", each part is a document and appears directly within the Body. The same mechanisms are used to define the contents of these two elements (Body or wrapper element).

```
<definitions…>
    <binding…>
        <operation…>
           <input>
               <soap:body parts="nmtokens"? use="literal|encoded"?
                          encodingStyle="uri-list"? namespace="uri"?>
           </input>
           <output>
               <soap:body parts="nmtokens"? use="literal|encoded"?
                          encodingStyle="uri-list"? namespace="uri"?>
           </output>
        </operation>
    </binding>
</definitions>
```

The optional **parts** attribute of type nmtokens indicates which parts appear somewhere within the SOAP Body portion of the message (other parts of a message may appear in other portions of the message such as when SOAP is used in conjunction with the multipart/related MIME binding). If the parts attribute is omitted, then all parts defined by the message are assumed to be included in the SOAP Body portion.

The required **use** attribute indicates whether the message parts are encoded using some encoding rules, or whether the parts defined the concrete schema of the message.

If use is **encoded**, then each message part references an abstract type using the **type** attribute. These abstract types are used to produce a concrete message by applying an encoding specified by the **encodingStyle** attribute. 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. If the referenced encoding style allows variations in it's format (such as the SOAP encoding does), then all variations MUST be supported ("reader makes right").

If use is **literal**, then each part references a concrete schema using the **element** attribute for simple parts or **type** attribute for composite parts (see section 2.3.1). The value of the **encodingStyle** attribute MAY be used to indicate that the concrete format was derived using a particular encoding (such as the SOAP encoding), but that only the specified variation is supported ("writer makes right").

The the value of the **encodingStyle** attribute is a list of URIs, each separated by a single space. The URI's represent encodings used within the message, in order from most restrictive to least restrictive (exactly like the encodingStyle attribute defined in the SOAP specification).

## 3.6 soap:fault

The soap:fault element specifies the contents of the contents of the SOAP Fault Details element. It is patterned after the soap:body element (see section 3.5).

```
<definitions…>
    <binding…>
        <operation…>
           <fault>*
               <soap:fault name="nmtoken" use="literal|encoded"
                                 encodingStyle="uri-list"? namespace="uri"?>
           </fault>
        </operation>
    </binding>
</definitions>
```

The **name** attribute relates the soap:fault to the wsdl:fault defined for the operation.

The fault message MUST have a single part. The **use**, **encodingStyle** and **namespace** attributes are all used in the same way as with soap:body (see section 3.5).

## 3.7 soap:header

The soap:header element allows headers to be defined that will be transmitted inside the Header element of the SOAP Envelope. It is not necessary to exhaustively list all headers in this section; it may be common that other specifications in the WSDL document cause headers to be added to the actual payload and it is not required to list those headers here.

```
<definitions…>
    <binding…>
        <operation…>
           <input>
               <soap:header element="qname" fault="qname"?>*
           </input>
           <output>
               <soap:header element="qname" fault="qname"?>*
           </output>
        </operation>
    </binding>
</definitions>
```

The **element** attribute (of type QName) references the concrete schema for the header element. The schema for a header MAY NOT include definitions for the soap:actor and soap:mustUnderstand attributes.

An optional **fault** attribute (similar to the element attribute) allow specification of the concrete schema for the header in case of an error pertaining to the header. The SOAP specification says that errors pertaining to headers must be returned in headers, and this mechanism allows specification of the schema for such headers.

## 3.8 soap:address

The SOAP address binding is used to give a port an address (a URI). A port using the SOAP binding MUST specify exactly one address.

```
<definitions…>
    <port…>
        <binding…>
           <soap:address location="uri"/>
        </binding>
    </port>
</definitions>
```

# 4. HTTP GET & POST Binding

WSDL includes a binding for HTTP 1.1"s GET and POST verbs in order to describe the interaction between a Web Browser and a web site. This allows applications other than Web Browsers to interact with the site. The following protocol specific information may be specified:

- An indication that a binding uses HTTP GET or POST
- An address for the port
- A relative address for each operation (relative to the base address defined by the port)

## 4.1 HTTP GET/POST Examples

The following example shows three ports that are bound differently for a given port type.

If the values being passed are part1=1, part2=2, part3=3, the request format would be as follows for each port:

```
port1: GET, URL="http://example.com/o1/A1B2/3"
port2: GET, URL="http://example.com/o1?p1=1&p2=2&p3=3
```

```
port3: POST, URL="http://example.com/o1", PAYLOAD="p1=1&p2=2&p3=3"
```

For each port, the response is either a GIF or a JPEG image.

**Example 5. GET and FORM POST returning GIF or JPG**

```
<definitions…>
    <message name="m1">
        <part name="part1" type="xsd:string"/>
        <part name="part2" type="xsd:int"/>
        <part name="part3" type="xsd:string"/>
    </message>

    <message name="m2">
        <part name="image" type="xsd:binary"/>
    </message>

    <portType name="pt1">
        <operation name="o1">
           <input message="tns:m1"/>
           <output message="tns:m2"/>
        </operation>
    </portType>

    <service name="service1">
        <port name="port1" binding="tns:b1">
           <http:address location="http://example.com/"/>
        </port>
        <port name="port2" binding="tns:b1">
           <http:address location="http://example.com/"/>
        </port>
        <port name="port3" binding="tns:b1">
             <http:address location="http://example.com/"/>
        </port>
    </service>

    <binding name="b1" type="pt1">
        <http:binding verb="GET"/>
        <operation name="o1">
           <http:operation location="o1/A(part1)B(part2)/(part3)"/>
           <input>
               <http:urlReplacement/>
           </input>
           <output>
               <mime:content type="image/gif"/>
               <mime:content type="image/jpeg"/>
           </output>
        </operation>
    </binding>

    <binding name="b2" type="pt1">
        <http:binding verb="GET"/>
        <operation name="o1">
           <http:operation location="o1"/>
           <input>
               <http:urlEncoded/>
           </input>
           <output>
               <mime:content type="image/gif"/>
               <mime:content type="image/jpeg"/>
           </output>
        </operation>
    </binding>

    <binding name="b3" type="pt1">
        <http:binding verb="POST"/>
        <operation name="o1">
           <http:operation location="o1"/>
           <input>
               <mime:content type="application/x-www-form-urlencoded"/>
           </input>
           <output>
               <mime:content type="image/gif"/>
```

```
                <mime:content type="image/jpeg"/>
            </output>
        </operation>
    </binding>
</definitions>
```

## 4.2 How the HTTP GET/POST Binding Extends WSDL

The HTTP GET/POST Binding extends WSDL with the following extension elements:

```
<definitions…>
    <binding…>
        <http:binding verb="nmtoken"/>
        <operation…>
           <http:operation location="uri"/>
           <input…>
               <-- mime elements -->
           </input>
           <output…>
               <-- mime elements -->
           </output>
        </operation>
    </binding>

    <port…>
        <http:address location="uri"/>
    </port>
</definitions>
```

These elements are covered in the subsequent sections.

## 4.3 http:address

The **location** attribute specifies the base URI for the port. The value of the attribute is combined with the values of the location attribute of the http:operation binding element. See section 4.5 for more details.

## 4.4 http:binding

The **http:binding** element indicates that this binding uses the HTTP protocol.

```
<definitions…>
    <binding…>
        <http:binding verb="nmtoken"/>
    </binding>
</definitions>
```

The value of the required **verb** attribute indicates the HTTP verb. Common values are GET or POST, but others may be used. Note that HTTP verbs are case sensitive.

## 4.5 http:operation

The **location** attribute specifies a relative URI for the operation. This URI is combined with the URI specified in the http:address element to form the full URI for the HTTP request. The URI value MUST be a relative URI.

```
<definitions…>
    <binding…>
        <operation…>
           <http:operation location="uri"/>
        </operation>
    </binding>
</definitions>
```

## 4.6 http:urlEncoded

The urlEncoded element indicates that all the message parts are encoded into the HTTP request URI using the standard URI-encoding rules (name1=value&name2=value…). The names of the parameters correspond to the names of the message parts. Each value contributed by the part is encoded using a name=value pair. This may be used with GET to specify URL encoding, or with POST to specify a FORM-POST. For GET, the "?" character is automatically appended as necessary.

```
<http:urlEncoded/>
```

For more information on the rules for URI-encoding parameters, see [5], [6], and [7].

## 4.7 http:urlReplacement

The **http:urlReplacment** element indicates that all the message parts are encoded into the HTTP request URI using a replacement algorithm:

- The relative URI value of http:operation is searched for a set of search patterns.
- The search occurs before the value of the http:operation is combined with the value of the location attribute from http:address.
- There is one search pattern for each message part. The search pattern string is the name of the message part surrounded with parenthesis "(" and ")".
- For each match, the value of the corresponding message part is substituted for the match at the location of the match.
- Matches are performed before any values are replaced (replaced values do not trigger additional matches).

Message parts MAY NOT have repeating values.

```
<http:urlReplacement/>
```

# 5. MIME Binding

WSDL includes a way to bind abstract types to concrete messages in some MIME format. Bindings for the following MIME types are defined:

- multipart/related
- text/xml
- application/x-www-form-urlencoded (the format used to submit a form in HTML)
- Others (by specifying the MIME type string)

The set of defined MIME types is both large and evolving, so it is not a goal for WSDL to exhaustively define XML grammar for each MIME type. Nothing precludes additional grammar to be added to define additional MIME types as necessary. If a MIME type string is sufficient to describe the content, the mime element defined below can be used.

## 5.11 MIME Binding example

**Example 7. Using multipart/related with SOAP**

This example describes that a GetCompanyInfo SOAP 1.1 request may be sent to a StockQuote service via the SOAP 1.1 HTTP binding. The request takes a ticker symbol of type string. The response contains multiple parts encoded in the MIME format multipart/related: a SOAP Envelope containing the current stock price as a float, zero or more marketing literature documents in HTML format, and an optional company logo in either GIF or JPEG format.

```
<definitions…>

    <types>
        <schema…>
           <element name="GetCompanyInfo">
               <complexType>
                   <all>
                       <element name="tickerSymbol " type="string"/>
                   </all>
               </complexType>
           </element>
           <element name="GetCompanyInfoResult">
               <complexType>
                   <all>
                       <element name="result" type="float"/>
                   </all>
               </complexType>
           </element>
           <complexType name="ArrayOfBinary" base="soap:Array">
```

```
                <all>
                    <element name="value" type="xsd:binary"/>
                </all>
            </complexType>
         </schema>
     </types>

     <message name="m1">
         <part name="body" element="tns:GetCompanyInfo"/>
     </message>

     <message name="m2">
         <part name="body" element="tns:GetCompanyInfoResult"/>
         <part name="docs" type="xsd:string"/>
         <part name="logo" type="tns:ArrayOfBinary"/>
     </message>

     <portType name="pt1">
         <operation name="GetCompanyInfo">
            <input message="m1"/>
            <output message="m2"/>
         </operation>
     </portType>

     <binding name="b1" type="tns:pt1">
         <operation name="GetCompanyInfo">
            <soap:operation soapAction="http://example.com/GetCompanyInfo"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <mime:multipartRelated>
                    <mime:part>
                        <soap:body parts="body" use="literal"/>
                    </mime:part>
                    <mime:part>
                        <mime:content part="docs" type="text/html"/>
                    </mime:part>
                    <mime:part>
                        <mime:content part="logo" type="image/gif"/>
                        <mime:content part="logo" type="image/jpeg"/>
                    </mime:part>
                </mime:multipartRelated>
            </output>
         </operation>
     </binding>

     <service name="CompanyInfoService">
         <port name="CompanyInfoPort"binding="tns:b1">
            <soap:address location="http://example.com/companyinfo"/>
         </port>
     </service>
</definitions>
```

## 5.2 How the MIME Binding extends WSDL

The MIME Binding extends WSDL with the following extension elements:

```
<mime:content part="nmtoken"? type="string"?/>

<mime:multipartRelated>
    <mime:part> *
        <-- mime element -->
    </mime:part>
</mime:multipartRelated>

<mime:mimeXml part="nmtoken"?/>
```

They are used at the following locations in WSDL:

```
<definitions…>
    <binding…>
        <operation…>
           <input…>
```

```
                <-- mime elements -->
            </input>
            <output…>
                <-- mime elements -->
            </output>
        </operation>
    </binding>
</definitions>
```

MIME elements appear under input and output to specify the MIME format. If multiple appear, they are considered to be alternatives.

## 5.3 mime:content

To avoid having to define a new element for every MIME format, the **mime:content** element may be used if there is no additional information to convey about the format other than its MIME type string.

```
<mime:content part="nmtoken"? type="string"?/>
```

The **part** attribute is used to specify the name of the message part. If the message has a single part, then the part attribute is optional. The **type** attribute contains the MIME type string. A type value has two portions, separated by a slash (/), either of which may be a wildcard (*). Not specifying the type attribute indicates that all MIME types are acceptable.

If the return format is XML, but the schema is not known ahead of time, the generic mime element can be used indicating text/xml:

```
<mime:content type="text/xml"/>
```

A wildcard (*) can be used to specify a family of mime types, for example all text types.

```
<mime:content type="text/*"/>
```

The following two examples both specify all mime types:

```
<mime:content type="*/*"/>
<mime:content/>
```

## 5.4 mime:multipartRelated

The multipart/related MIME type aggregates an arbitrary set of MIME formatted parts into one message using the MIME type "multipart/related". The **mime:multipartRelated** element describes the concrete format of such a message:

```
<mime:multipartRelated>
    <mime:part> *
        <-- mime element -->
    </mime:part>
</mime:multipartRelated>
```

The **mime:part** element describes each part of a multipart/related message. MIME elements appear within **mime:part** to specify the concrete MIME type for the part. If more than one MIME element appears inside a mime:part, they are alternatives.

## 5.5 soap:body

When using the MIME binding with SOAP requests, it is legal to use the soap:body element as a MIME element. It indicates the content type is "text/xml", and there is an enclosing SOAP Envelope.

## 5.6 mime:mimeXml

To specify XML payloads that are not SOAP compliant (do not have a SOAP Envelope), but do have a particular schema, the **mime:mimeXml** element may be used to specify that concrete schema. The **part** attribute refers to a message part defining the concrete schema of the root XML element. The part attribute MAY be omitted if the message has only a single part. The part references a concrete schema using the **element** attribute for simple parts or **type** attribute for composite parts (see section 2.3.1).

```
<mime:mimeXml part="nmtoken"?/>
```

# 6. References

[2] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997

[4] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[5] http://www.w3.org/TR/html401/interact/forms.html - submit-format

[6] http://www.w3.org/TR/html401/appendix/notes.html - ampersands-in-uris

[7] http://www.w3.org/TR/html401/interact/forms.html - h-17.13.4

[10] W3C Working Draft "XML Schema Part 1: Structures". This is work in progress.

[11] W3C Working Draft "XML Schema Part 2: Datatypes". This is work in progress.

# A 1. Notes on URIs

This section does not directly contribute to the specification, but provide background that may be useful when implementing the specification.

## A 1.1 XML namespaces & schema locations

It is a common misperception to equate the targetNamespace of an XML schema or the value of the *xmlns* attribute in XML instances with the location of the corresponding schema. Since namespaces are in fact URIs, and URIs may be locations, and you may be able to retrieve a schema from that location, it does not mean that is the only schema that is associated with that namespace. There can be multiple schemas associated with a particular namespace, and it is up to a processor of XML to determine which one to use in a particular processing context. The WSDL specification provides the processing context here via the <*import*> mechanism, which is based on the XML schemas grammar for the similar concept.

## A 1.2 Relative URIs

Throughout this document you see fully qualified URIs used in WSDL and XSD documents. The use of a fully qualified URI is simply to illustrate the referencing concepts. The use of relative URIs is completely allowed and is warranted in many cases. For information on processing relative URIs, see http://www.normos.org/ietf/rfc/rfc2396.txt.

## A 1.3 Generating URIs

When working with WSDL, it is sometimes desirable to make up a URI for an entity, but not make the URI globally unique for all time and have it "mean" that version of the entity (schema, WSDL document, etc.). There is a particular URI base reserved for use for this type of behavior. The base URI http://tempuri.org/can be used to construct a URI without any unique association to an entity. For example, two people or programs could choose to simultaneously use the URI http://tempuri.org/myschemafor two completely different schemas, and as long as the scope of the use of the URIs does not intersect, then they are considered unique enough. This has the further benefit that the entity referred to by the URI can be versioned without having to generate a new URI, as long as it makes sense within the processing context. It is not recommended that http://tempuri.org/be used as a base for stable, fixed entities.

# A 2. Wire format for WSDL examples

## A 2.1. Example 1

SOAP Message Embedded in HTTP Request

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
```

```
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Embedded in HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## A 3. Location of Extensibility Elements

Extensibility elements can appear at the following locations in a WSDL document:

| Location | Meaning | Possible usage |
|---|---|---|
| definitions | The extensibility element applies to the WSDL document as a whole. | · Introduce additional information or definitions to a WSDL document as a whole. |
| definitions/types | The extensibility element is a type system. | · Specify the format of the message in a type system other than XSD. |
| definitions/service | The extensibility element applies to the service. | · Introduce additional information or definitions for the service. |
| definitions/service/port | The extensibility element applies to the port. | · Specify an address for the port. |
| definitions/binding | The extensibility element applies to the binding as a whole. | · Provide protocol specific information that applies to all the operations in the port type being bound. |
| definitions/binding/operation | The extensibility element applies to the operation as a whole. | · Provide protocol specific information that applies to both the input message and the output message. |
| definitions/binding/operation/input | The extensibility element applies to the input message for the operation. | · Provide details on how abstract message parts map into the concrete protocol and data formats of the binding.<br><br>· Provide additional protocol specific information for the input message. |

| | | |
|---|---|---|
| definitions/binding/operation/output | The extensibility element applies to the output message of the operation. | · Provide details on how abstract message parts map into the concrete protocol and data formats of the binding.<br><br>· Provide additional protocol specific information for the output message. |
| definitions/binding/operation/fault | The extensibility element applies to a fault message of the operation. | · Provide details on how abstract message parts map into the concrete protocol and data formats of the binding.<br><br>· Provide additional protocol specific information for the fault message. |

## A 4. Schemas

### A 4.1 WSDL Schema

```
<schema  targetNamespace="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://www.w3.org/1999/XMLSchema"
   elementFormDefault="qualified">
   <element name="documentation" >
      <complexType content="mixed">
         <any minOccurs="0" maxOccurs="unbounded"/>
         <anyAttribute/>
      </complexType>
   </element>
   <complexType name="documented"  abstract="true" content="elementOnly">
      <element ref="wsdl:documentation"/>
   </complexType>
   <complexType name="openAtts" abstract="true" content="elementOnly">
      <annotation>
         <documentation>
         This type is extended by  component types
         to allow attributes from other namespaces to be added.
         </documentation>
      </annotation>
      <element ref="wsdl:documentation"/>
      <anyAttribute namespace="##other"/>
   </complexType>
   <element name="definitions" type="wsdl:definitionsType">
      <key name="message">
         <selector>message</selector>
         <field>@name</field>
      </key>
      <key name="portType">
         <selector>portType</selector>
         <field>@name</field>
      </key>
      <key name="binding">
         <selector>binding</selector>
         <field>@name</field>
      </key>
      <key name="service">
         <selector>service</selector>
         <field>@name</field>
      </key>
      <key name="import">
         <!-- Is it too restrictive?-->
            <selector>import</selector>
            <field>@namespace</field>
         </key>
      <key name="port">
         <selector>service/port</selector>
         <field>@name</field>
      </key>
   </element>
   <complexType name="definitionsType" base="wsdl:documented" derivedBy="extension">
      <element ref="wsdl:import" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="wsdl:types" minOccurs="0" maxOccurs="1"/>
      <element ref="wsdl:message" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="wsdl:portType" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="wsdl:binding" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="wsdl:service" minOccurs="0" maxOccurs="unbounded"/>
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded">
         <annotation>
            <documentation>to support extensibility elements </documentation>
          </annotation>
      </any>
      <attribute name="targetNamespace" type="uriReference" use="optional"/>
      <attribute name="name" type="NMTOKEN" use="optional"/>
   </complexType>
   <element name="import" type="wsdl:importType" />
   <complexType name="importType" base="wsdl:documented" derivedBy="extension" >
      <attribute name="namespace" type="uriReference"  use="required"/>
      <attribute name="location" type="uriReference" use="required"/>
   </complexType>
   <element name="types" type="wsdl:typesType"/>
   <complexType name="typesType" base="wsdl:documented" derivedBy="extension" >
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </complexType>
   <element name="message" type="wsdl:messageType">
      <unique name="part">
         <selector>part</selector>
         <field>@name</field>
      </unique>
   </element>
   <complexType name="messageType" base="wsdl:documented" derivedBy="extension">
      <element ref="wsdl:part" minOccurs="0" maxOccurs="unbounded"/>
      <attribute name="name" type="NCName" use="required"/>
   </complexType>
   <element name="part" type="wsdl:partType"/>
   <complexType name="partType" base="wsdl:openAtts" derivedBy="extension">
      <attribute name="name" type="NMTOKEN" use="optional"/>
      <attribute name="type" type="QName" use="optional"/>
      <attribute name="element" type="QName" use="optional"/>
   </complexType>
   <element name="portType" type="wsdl:portTypeType">
      <key name="operation">
         <selector>operation</selector>
         <field>@name</field>
      </key>
   </element>
   <complexType name="portTypeType" base="wsdl:documented" derivedBy="extension"  content="elementOnly" >
      <element ref="wsdl:operation" minOccurs="0" maxOccurs="unbounded"/>
      <attribute name="name" type="NCName" use="required"/>
   </complexType>
   <element name="operation" type="wsdl:operationType">
      <unique name="paramName">
         <selector>input| output| fault</selector>
         <field>@name</field>
      </unique>
   </element>
   <complexType name="operationType" base="wsdl:documented" derivedBy="extension" >
      <choice minOccurs="1" maxOccurs="1">
         <group ref="wsdl:one-way-operation"/>
         <group ref="wsdl:request-response-operation"/>
         <group ref="wsdl:solicit-response-operation"/>
         <group ref="wsdl:notification-operation"/>
      </choice>
      <attribute name="name" type="NCName" use="required"/>
   </complexType>
   <group name="one-way-operation">
      <sequence>
         <element ref="wsdl:input" />
      </sequence>
   </group>
   <group name="request-response-operation">
      <sequence>
         <element ref="wsdl:input"/>
         <element ref="wsdl:output"/>
         <element ref="wsdl:fault"  minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </group>
   <group name="solicit-response-operation">
      <sequence>
         <element ref="wsdl:output"/>
         <element ref="wsdl:input" />
         <element ref="wsdl:fault"  minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
   </group>
   <group name="notification-operation">
      <sequence>
         <element ref="wsdl:output"/>
      </sequence>
   </group>
   <element name="input"  type="wsdl:paramType"/>
   <element name="output" type="wsdl:paramType"/>
   <element name="fault" type="wsdl:faultType"/>
   <complexType name="paramType" base="wsdl:documented" derivedBy="extension" >
      <attribute name="name" type="NMTOKEN" use="optional"/>
      <attribute name="message" type="QName" use="required"/>
   </complexType>
   <complexType name="faultType" base="wsdl:documented" derivedBy="extension" >
      <attribute name="name" type="NMTOKEN" use="required"/>
      <attribute name="message" type="QName" use="required"/>
   </complexType>
   <complexType name="startWithExtensionsType" base="wsdl:documented" derivedBy="extension"
      content="elementOnly" abstract="true">
      <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
   </complexType>
   <element name="binding"  type="wsdl:bindingType"/>
   <complexType name="bindingType" base="wsdl:startWithExtensionsType" derivedBy="extension"
      content="elementOnly">
      <element name="operation" type="wsdl:binding_operationType" minOccurs="0" maxOccurs="unbounded"/>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="type"  type="QName" use="required"/>
   </complexType>
   <complexType name="binding_operationType" base="wsdl:startWithExtensionsType"
      derivedBy="extension" content="elementOnly">
      <element name="input"  type="wsdl:startWithExtensionsType" minOccurs="0" maxOccurs="1"/>
      <element name="output"  type="wsdl:startWithExtensionsType" minOccurs="0" maxOccurs="1"/>
      <element name="fault" minOccurs="0" maxOccurs="unbounded">
         <complexType base="wsdl:startWithExtensionsType" derivedBy="extension">
            <attribute name="name" type="NMTOKEN" use="required"/>
         </complexType>
      </element>
      <attribute name="name" type="NCName" use="required"/>
   </complexType>
   <element name="service" type="wsdl:serviceType"/>
   <complexType name="serviceType" base="wsdl:documented" derivedBy="extension" >
      <element ref="wsdl:port" minOccurs="0" maxOccurs="unbounded"/>
      <any namespace="##other" minOccurs="0" maxOccurs="1"/>
      <attribute name="name" type="NCName" use="required"/>
   </complexType>
   <element name="port" type="wsdl:portType"/>
   <complexType name="portType"  base="wsdl:documented" derivedBy="extension">
      <any namespace="##other" minOccurs="0" maxOccurs="1"/>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="binding" type="QName" use="required"/>
   </complexType>
</schema>
```

## A 4.2 SOAP Binding Schema

```
<schema  targetNamespace="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://www.w3.org/1999/XMLSchema">
   <element name="binding" type="soap:bindingType"/>
   <complexType name="bindingType" content="empty">
      <attribute name="transport" type="uriReference" use="optional"/>
      <attribute name="style" type="soap:styleChoice" use="optional"/>
   </complexType>
   <simpleType name="styleChoice" base="string" derivedBy="restriction">
      <enumeration value="rpc"/>
      <enumeration value="document"/>
   </simpleType>
   <element name="operation" type="soap:operationType"/>
   <complexType name="operationType" content="empty">
      <attribute name="soapAction" type="uriReference" use="optional"/>
      <attribute name="style" type="soap:styleChoice" use="optional"/>
   </complexType>
   <element name="body" type="soap:bodyType"/>
   <complexType name="bodyType" content="empty">
      <attribute name="encodingStyle" type="uriReference" use="optional"/>
      <attribute name="parts" type="NMTOKENS" use="optional"/>
      <attribute name="use" type="soap:useChoice" use="optional"/>
      <attribute name="namespace" type="uriReference" use="optional"/>
   </complexType>
   <simpleType name="useChoice" base="string">
      <enumeration value="literal"/>
      <enumeration value="encoded"/>
   </simpleType>
   <element name="fault" type="soap:faultType"/>
   <complexType name="faultType" base="soap:bodyType" derivedBy="restriction">
      <attribute name="parts" type="NMTOKENS"  use="prohibited"/>
   </complexType>
   <element name="header" type="soap:headerType"/>
   <complexType name="headerType" content="empty">
      <attribute name="element" type="QName" use="required"/>
      <attribute name="fault" type="QName" use="optional"/>
   </complexType>
   <element name="address" type="soap:addressType"/>
   <complexType name="addressType" content="empty">
      <attribute name="location" type="uriReference" use="required"/>
   </complexType>
</schema>
```

## A 4.3 HTTP Binding Schema

```
<schema  targetNamespace="http://schemas.xmlsoap.org/wsdl/http/"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   xmlns="http://www.w3.org/1999/XMLSchema">
   <element name="address" type="http:addressType"/>
   <complexType name="addressType" content="empty">
      <attribute name="location" type="uriReference" use="required"/>
   </complexType>
   <element name="binding" type="http:bindingType"/>
   <complexType name="bindingType" content="empty">
      <attribute name="verb" type="NMTOKEN" use="required"/>
   </complexType>
   <element name="operation" type="http:operationType"/>
   <complexType name="operationType" content="empty">
      <attribute name="location" type="uriReference" use="required"/>
   </complexType>
   <element name="urlEncoded">
      <complexType content="empty"/>
   </element>
   <element name="urlReplacement">
      <complexType content="empty"/>
   </element>
</schema>
```

## A 4.4 MIME Binding Schema

```
<schema  targetNamespace="http://schemas.xmlsoap.org/wsdl/mime/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
```

```
    xmlns="http://www.w3.org/1999/XMLSchema">
    <element name="content" type="mime:contentType"/>
    <complexType name="contentType" content="empty">
       <attribute name="type" type="string" use="optional"/>
       <attribute name="part" type="NMTOKEN" use="optional"/>
    </complexType>
    <element name="multipartRelated" type="mime:multipartRelatedType"/>
    <complexType name="multipartRelatedType" content="elementOnly">
       <element ref="mime:part" minOccurs="0" maxOccurs="unbounded"/>
    </complexType>
    <element name="part" type="mime:partType"/>
    <complexType name="partType" content="elementOnly">
       <any namespace="targetNamespace" minOccurs="0" maxOccurs="unbounded"/>
       <attribute name="name" type="NMTOKEN" use="required"/>
    </complexType>
    <element name="mimeXml" type="mime:mimeXmlType"/>
    <complexType name="mimeXmlType" content="empty">
       <attribute name="part" type="NMTOKEN" use="optional"/>
    </complexType>
 </schema>
```

**What do you think of this article?**

○ Killer! (5) ○ Good stuff (4) ○ So-so; not bad (3) ○ Needs work (2) ○ Lame! (1)

**Comments?**

Submit feedback

Privacy | Legal | Contact