Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language

W3C Working Draft 3 August 2004

This version:
http://www.w3.org/TR/2004/WD-wsdl20-20040803
Latest version:
http://www.w3.org/TR/wsdl20
Previous versions:
http://www.w3.org/TR/2004/WD-wsdl20-20040326
Editors:
Roberto Chinnici, Sun Microsystems
Martin Gudgin, Microsoft
Jean-Jacques Moreau, Canon
Jeffrey Schlimmer, Microsoft
Sanjiva Weerawarana, IBM Research

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

Copyright Â© 2004 W3CÂ® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This document describes the Web Services Description Language (WSDL) Version 2.0, an XML language for describing Web services. This specification defines the core language which can be used to describe Web services based on an abstract model of what the service offers. It also defines criteria for a conformant processor of this language.

Status of this Document

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

This is a W3C Last Call Working Draft. If the feedback is positive, the Working Group plans to submit this specification for consideration as a W3C Candidate Recommendation. Comments on this document are invited and are to be sent to the public public-ws-desc-comments@w3.org mailing list (public

archive). Comments can be sent until **4 October 2004**.

Three formal objections from Working Group participants have been received against portions of the WSDL 2.0 specification. Feedback is specifically encouraged on these topics:

- Compositors (see objection)
- Feature and properties (see objection and follow-on message)
- Requiring unique GEDs or required feature to distinguish operations (see objection)

A diff-marked version against the previous version of this document is available. For a detailed list of changes since the last publication of this document, please refer to appendix **F. Part 1 Change Log** [p.96] . Issues about this document are documented in the last call issues list maintained by the Working Group.

This document has been produced as part of the W3C Web Services Activity. The authors of this document are the Web Services Description Working Group members.

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

This document has been produced under the 24 January 2002 Current Patent Practice as amended by the W3C Patent Policy Transition Procedure. Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Short Table of Contents

1. Introduction [p.7]
2. Component Model [p.9]
3. Types [p.68]
4. Modularizing WSDL descriptions [p.73]
5. Documentation [p.77]
6. Language Extensibility [p.77]
7. Locating WSDL Documents [p.80]
8. Conformance [p.80]
9. XML Syntax Summary (Non-Normative) [p.82]
10. References [p.84]
A. The application/wsdl+xml Media Type [p.87]
B. Acknowledgements [p.89] (Non-Normative)
C. URI References for WSDL constructs [p.90] (Non-Normative)
D. Migrating from WSDL 1.1 to WSDL 2.0 [p.92] (Non-Normative)
E. Examples of Specifications of Extension Elements for Alternative Schema Language Support. [p.93] (Non-Normative)
F. Part 1 Change Log [p.96] (Non-Normative)

Table of Contents

1. Introduction [p.7]
  1.1 Web Service [p.7]
  1.2 Notational Conventions [p.7]
  1.3 WSDL Terminology [p.9]
2. Component Model [p.9]
  2.1 Definitions [p.11]
    2.1.1 The Definitions Component [p.11]
    2.1.2 XML Representation of Definitions Component [p.12]
      2.1.2.1 targetNamespace attribute information item [p.13]
    2.1.3 Mapping Definitions' XML Representation to Component Properties [p.13]
  2.2 Interface [p.14]
    2.2.1 The Interface Component [p.14]
      2.2.1.1 Operation Name Mapping Requirement [p.15]
    2.2.2 XML Representation of Interface Component [p.16]
      2.2.2.1 name attribute information item with interface [owner] [p.17]
      2.2.2.2 extends attribute information item [p.17]
      2.2.2.3 styleDefault attribute information item [p.17]
    2.2.3 Mapping Interface's XML Representation to Component Properties [p.18]
  2.3 Interface Fault [p.19]
    2.3.1 The Interface Fault Component [p.19]
    2.3.2 XML Representation of Interface Fault Component [p.20]
      2.3.2.1 name attribute information item with fault [owner] [p.21]
      2.3.2.2 element attribute information item with fault [owner] [p.21]
    2.3.3 Mapping Interface Fault's XML Representation to Component Properties [p.22]
  2.4 Interface Operation [p.22]
    2.4.1 The Interface Operation Component [p.22]
      2.4.1.1 Operation Style [p.24]
    2.4.2 RPC Style [p.24]
      2.4.2.1 wrpc:signature Extension [p.25]
      2.4.2.2 XML Representation of the wrpc:signature Extension [p.27]
      2.4.2.3 wrpc:signature Extension Mapping To Properties of an Interface Operation Component [p.27]
    2.4.3 XML Representation of Interface Operation Component [p.28]
      2.4.3.1 name attribute information item with operation [owner] [p.29]
      2.4.3.2 pattern attribute information item with operation [owner] [p.30]
      2.4.3.3 style attribute information item with operation [owner] [p.30]
      2.4.3.4 safe attribute information item with operation [owner] [p.30]
    2.4.4 Mapping Interface Operation's XML Representation to Component Properties [p.30]
  2.5 Message Reference [p.31]
    2.5.1 The Message Reference Component [p.31]
    2.5.2 XML Representation of Message Reference Component [p.32]
      2.5.2.1 messageLabel attribute information item with input, or output [owner] [p.33]
      2.5.2.2 element attribute information item with input, or output [owner] [p.34]

2.5.3 Mapping Message Reference's XML Representation to Component Properties [p.34]
2.6 Fault Reference [p.35]
2.6.1 The Fault Reference Component [p.35]
2.6.2 XML Representation of Fault Reference Component [p.36]
2.6.2.1 ref attribute information item with infault, or outfault [owner] [p.37]
2.6.2.2 messageLabel attribute information item with infault, or outfault [owner] [p.38]
2.6.3 Mapping Fault Reference's XML Representation to Component Properties [p.38]
2.7 Feature [p.39]
2.7.1 The Feature Component [p.39]
2.7.1.1 Feature Composition Model [p.39]
2.7.1.1.1 Example of Feature Composition Model [p.40]
2.7.2 XML Representation of Feature Component [p.41]
2.7.2.1 uri attribute information item with feature [owner] [p.41]
2.7.2.2 required attribute information item with feature [owner] [p.42]
2.7.3 Mapping Feature's XML Representation to Component Properties [p.42]
2.8 Property [p.42]
2.8.1 The Property Component [p.42]
2.8.1.1 Property Composition Model [p.43]
2.8.2 XML Representation of Property Component [p.44]
2.8.2.1 uri attribute information item with property [owner] [p.45]
2.8.2.2 required attribute information item with property [owner] [p.45]
2.8.2.3 value element information item with property [parent] [p.45]
2.8.2.4 constraint element information item with property [parent] [p.46]
2.8.3 Mapping Property's XML Representation to Component Properties [p.46]
2.9 Binding [p.47]
2.9.1 The Binding Component [p.47]
2.9.2 XML Representation of Binding Component [p.48]
2.9.2.1 name attribute information item with binding [owner] [p.49]
2.9.2.2 interface attribute information item with binding [owner] [p.49]
2.9.2.3 type attribute information item with binding [owner] [p.50]
2.9.2.4 Binding extension elements [p.50]
2.9.3 Mapping Binding's XML Representation to Component Properties [p.50]
2.10 Binding Fault [p.51]
2.10.1 The Binding Fault Component [p.51]
2.10.2 XML Representation of Binding Fault Component [p.52]
2.10.2.1 ref attribute information item with fault [owner] [p.53]
2.10.2.2 Binding Fault extension elements [p.53]
2.10.3 Mapping Binding Fault's XML Representation to Component Properties [p.53]
2.11 Binding Operation [p.53]
2.11.1 The Binding Operation Component [p.53]
2.11.2 XML Representation of Binding Operation Component [p.54]
2.11.2.1 ref attribute information item with operation [owner] [p.55]
2.11.2.2 Binding Operation extension elements [p.55]
2.11.3 Mapping Binding Operation's XML Representation to Component Properties [p.55]
2.12 Binding Message Reference [p.56]
2.12.1 The Binding Message Reference Component [p.56]
2.12.2 XML Representation of Binding Message Reference Component [p.57]

2.12.2.1 messageLabel attribute information item with input or output [owner] [p.58]
2.12.2.2 Binding Message Reference extension elements [p.58]
2.12.3 Mapping Binding Message Reference's XML Representation to Component Properties [p.58]
2.13 Service [p.59]
2.13.1 The Service Component [p.59]
2.13.2 XML Representation of Service Component [p.60]
2.13.2.1 name attribute information item with service [owner] [p.61]
2.13.2.2 interface attribute information item with service [owner] [p.61]
2.13.3 Mapping Service's XML Representation to Component Properties [p.61]
2.14 Endpoint [p.62]
2.14.1 The Endpoint Component [p.62]
2.14.2 XML Representation of Endpoint Component [p.63]
2.14.2.1 name attribute information item with endpoint [owner] [p.64]
2.14.2.2 binding attribute information item with endpoint [owner] [p.64]
2.14.2.3 address attribute information item with endpoint [owner] [p.64]
2.14.2.4 Endpoint extension elements [p.64]
2.14.3 Mapping Endpoint's XML Representation to Component Properties [p.65]
2.15 Definition of the Simple Types Used in the Component Model [p.65]
2.15.1 string Type [p.66]
2.15.2 Token Type [p.66]
2.15.3 NCName Type [p.66]
2.15.4 anyURI Type [p.66]
2.15.5 QName Type [p.66]
2.15.6 boolean Type [p.66]
2.15.7 int Type [p.66]
2.16 Equivalence of Components [p.67]
2.17 Symbol Spaces [p.67]
2.18 QName resolution [p.68]
2.19 Comparing URIs [p.68]
3. Types [p.68]
3.1 Using W3C XML Schema Description Language [p.70]
3.1.1 Importing XML Schema [p.70]
3.1.1.1 namespace attribute information item [p.70]
3.1.1.2 schemaLocation attribute information item [p.71]
3.1.2 Embedding XML Schema [p.71]
3.1.2.1 targetNamespace attribute information item [p.72]
3.1.3 References to Element Declarations and Type Definitions [p.72]
3.2 Using Other Schema Languages [p.72]
4. Modularizing WSDL descriptions [p.73]
4.1 Including Descriptions [p.73]
4.1.1 location attribute information item with include [owner] [p.74]
4.2 Importing Descriptions [p.75]
4.2.1 namespace attribute information item [p.76]
4.2.2 location attribute information item with import [owner] [p.77]
5. Documentation [p.77]
6. Language Extensibility [p.77]
6.1 Element based Extensibility [p.78]

6.1.1 Mandatory extensions [p.78]
6.1.2 required attribute information item [p.79]
6.2 Attribute-based Extensibility [p.79]
6.3 Extensibility Semantics [p.79]
7. Locating WSDL Documents [p.80]
7.1 wsdli:wsdlLocation attribute information item [p.80]
8. Conformance [p.80]
8.1 Document Conformance [p.80]
8.2 XML Information Set Conformance [p.81]
8.3 Processor Conformance [p.81]
9. XML Syntax Summary (Non-Normative) [p.82]
10. References [p.84]
10.1 Normative References [p.84]
10.2 Informative References [p.86]

Appendices

A. The application/wsdl+xml Media Type [p.87]
A.1 Registration [p.88]
A.2 Security considerations [p.89]
B. Acknowledgements [p.89] (Non-Normative)
C. URI References for WSDL constructs [p.90] (Non-Normative)
C.1 WSDL URIs [p.90]
C.2 Fragment Identifiers [p.91]
C.3 Extension Elements [p.91]
C.4 Example [p.92]
D. Migrating from WSDL 1.1 to WSDL 2.0 [p.92] (Non-Normative)
D.1 Operation Overloading [p.92]
D.2 PortTypes [p.93]
D.3 Ports [p.93]
E. Examples of Specifications of Extension Elements for Alternative Schema Language Support. [p.93] (Non-Normative)
E.1 DTD [p.93]
E.1.1 namespace attribute information item [p.93]
E.1.2 location attribute information item [p.94]
E.1.3 References to Element Definitions [p.94]
E.2 RELAX NG [p.94]
E.2.1 Importing RELAX NG [p.94]
E.2.1.1 ns attribute information item [p.95]
E.2.1.2 href attribute information item [p.95]
E.2.2 Embedding RELAX NG [p.95]
E.2.2.1 ns attribute information item [p.96]
E.2.3 References to Element Declarations [p.96]
F. Part 1 Change Log [p.96] (Non-Normative)
F.1 WSDL Specification Changes [p.96]

1. Introduction

Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

This specification defines a language for describing the abstract functionality of a service as well as a framework for describing the concrete details of a service description. It also defines criteria for a conformant processor of this language. The *WSDL Version 2.0 Part 2: Message Exchange Patterns* specification [*WSDL 2.0 Predefined Extensions [p.85]*] defines the sequence and cardinality of abstract messages sent or received by an operation. The *WSDL Version 2.0 Part 3: Bindings* specification [*WSDL 2.0 Bindings [p.85]*] defines a language for describing such concrete details for SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework [p.86]*], HTTP [*IETF RFC 2616 [p.86]*] and MIME [*IETF RFC 2045 [p.86]*].

1.1 Web Service

WSDL describes a Web service in two fundamental stages: one abstract and one concrete. Within each stage, the description uses a number of constructs to promote reusability of the description and separate independent design concerns.

At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are described independent of a specific wire format using a type system, typically XML Schema.

An *operation* associates a message exchange pattern with one or more messages. A *message exchange pattern* identifies the sequence and cardinality of messages sent and/or received as well as who they are logically sent to and/or received from. An *interface* groups together operations without any commitment to transport or wire format.

At a concrete level, a *binding* specifies transport and wire format details for one or more interfaces. An *endpoint* associates a network address with a binding. And finally, a *service* groups together endpoints that implement a common interface.

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 RFC 2119 [*IETF RFC 2119 [p.84]*].

This specification uses properties from the XML Information Set [*XML Information Set [p.85]*]. Such properties are denoted by square brackets, e.g. [namespace name].

This specification uses namespace prefixes throughout; they are listed in Table 1-1 [p.8] . Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [*XML Information Set [p.85]*]).

This specification uses curly brackets (e.g., {property}) to indicate a property in the WSDL component model, as defined in **2. Component Model** [p.9] .

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

Prefix	Namespace	Notes
wsdl	"http://www.w3.org/2004/08/wsdl"	A normative XML Schema [*XML Schema: Structures [p.85]*], [*XML Schema: Datatypes [p.85]*] document for the "http://www.w3.org/2004/08/wsdl" namespace can be found at http://www.w3.org/2004/08/wsdl. WSDL documents that do NOT conform to this schema are not valid WSDL documents. WSDL documents that DO conform to this schema and also conform to the other constraints defined in this specification are valid WSDL documents.
wsdli	"http://www.w3.org/2004/08/wsdl-instance"	A normative XML Schema [*XML Schema: Structures [p.85]*], [*XML Schema: Datatypes [p.85]*] document for the "http://www.w3.org/2004/08/wsdl-instance" namespace can be found at http://www.w3.org/2004/08/wsdl-instance.
wsdls	"http://www.w3.org/2004/08/wsdl-simple-types"	This prefix and namespace name are used to refer to the simple types defined by this specification for use in the component model, see **2.15 Definition of the Simple Types Used in the Component Model** [p.65] .
wrpc	"http://www.w3.org/2004/08/wsdl/rpc"	A normative XML Schema [*XML Schema: Structures [p.85]*], [*XML Schema: Datatypes [p.85]*] document for the "http://www.w3.org/2004/08/wsdl/rpc" namespace can be found at http://www.w3.org/2004/08/wsdl/rpc. WSDL documents that do NOT conform to this schema are not valid WSDL documents. WSDL documents that DO conform to this schema and also conform to the other constraints defined in this specification are valid WSDL documents.

wsoap	"http://www.w3.org/2004/08/wsdl/soap12"	Defined by WSDL 2.0: Bindings [*WSDL 2.0 Bindings [p.85]*].
whttp	"http://www.w3.org/2004/08/wsdl/http"	
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [*XML Schema: Structures [p.85]*], [*XML Schema: Datatypes [p.85]*].
xsi	"http://www.w3.org/2001/XMLSchema-instance"	

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [*IETF RFC 2396 [p.84]*].

All parts of this specification are normative, with the EXCEPTION of notes, pseudo-schemas, examples, and sections explicitly marked as "Non-Normative".

Pseudo-schemas are provided for each component, before the description of the component. They use BNF-style conventions for attributes and elements: *'?'* denotes optionality (i.e. zero or one occurrences), *'*'* denotes zero or more occurrences, *'+'* one or more occurrences, *'['* and *']'* are used to form groups, *'|'* represents choice. Attributes are conventionally assigned a value which corresponds to their type, as defined in the normative schema.

```
<!-- sample pseudo-schema -->
<defined_element
      required_attribute_of_type_string="xs:string"
      optional_attribute_of_type_int="xs:int"? >
  <required_element />
  <optional_element />?
  <one_or_more_of_these_elements />+
  [ <choice_1 /> | <choice_2 /> ]*
</defined_element>
```

1.3 WSDL Terminology

This section describes the terms and concepts introduced in Part 1 of the WSDL Version 2.0 specification (this document).

Actual Value

> As in [*XML Schema: Structures [p.85]*], the phrase actual value is used to refer to the member of the value space of the simple type definition associated with an attribute information item which corresponds to its normalized value. This will often be a string, but may also be an integer, a boolean, a URI reference, etc.

2. Component Model

This section describes the conceptual model of WSDL as a set of components with attached properties, which collectively describe a Web service. Each subsection herein describes a different type of component, its defined properties, and its representation as an XML Infoset [*XML Information Set [p.85]*].

Components are typed collections of properties that correspond to different aspects of Web services.

Properties are unordered and unique with respect to the component they are associated with. Individual properties' definitions may constrain their content (e.g., to a typed value, another component, or a set of typed values or components), and components may require the presence of a property to be considered conformant. Such properties are marked as REQUIRED, whereas those that are not required to be present are marked as OPTIONAL. By convention, when specifying the mapping rules from the XML Infoset representation of a component to the component itself, an optional property that is absent in the component in question is described as being "empty". Unless otherwise specified, when a property is identified as being a collection (a set or a list), its value may be a 0-element (empty) collection. In order to simplify the presentation of the rules that deal with sets of components, for all OPTIONAL properties whose type is a set, the absence of such a property from a component MUST be treated as semantically equivalent to the presence of a property with the same name and whose value is the empty set. In other words, every OPTIONAL set-valued property MUST be assumed to have the empty set as its default value, to be used in case the property is absent.

Component definitions are independent of any particular serialization of the component model. In order to avoid creating an implicit dependency on a particular serialization, this specification defines its own set of simple types for use by component definitions, rather than reusing an existing one (say [*XML Schema: Datatypes [p.85]*]). By convention, those types are defined in the http://www.w3.org/2004/08/wsdl-simple-types namespace and references to them use the *wsdls* prefix, see **2.15 Definition of the Simple Types Used in the Component Model** [p.65] . All the value spaces of all simple types used by the the component model are a superset of the value spaces of the XML Schema simple types with the same name, i.e. every *xs:string* is also a *wsdls:string* (but the opposite is not true). Hence, for brevity, in the sections describing the mapping from the XML Infoset representation of a WSDL document to its component model we use "actual values" as defined by the XML Schema specification [*XML Schema: Datatypes [p.85]*] as if they were members of the value space of the corresponding WSDL-defined simple types. So, for instance, we talk of "assigning the actual value of the "name" *attribute information item* (a *xs:string*) "to the {name} property (of type *wsdls:string*)" of a certain component.

In addition to the direct XML Infoset representation described here, the component model allows components external to the Infoset through the mechanisms described in **4. Modularizing WSDL descriptions** [p.73] .

A component model can be extracted from a given XML Infoset which conforms to the XML Schema for WSDL by recursively mapping Information Items to their identified components, starting with the wsdl:description element information item. This includes the application of the mechanisms described in **4. Modularizing WSDL descriptions** [p.73] .

This document does not specify a means of producing an XML Infoset representation from a given set of WSDL components. Furthermore, given a particular serialization, not all valid sets of components need be serializable to it. For instance, due to the use in the component model of types that cannot be described using XML schema (.e.g *wsdls:string*), it is possible to come up with a valid set of WSDL components that cannot be serialized as an XML 1.0 document.

2.1 Definitions

2.1.1 The Definitions Component

At the abstract level, the Definitions component is just a container for two categories of components; WSDL components and type system components.

WSDL components are interfaces, bindings and services.

Type system components describe the constraints on a message's content. By default, these constraints are expressed in terms of the [*XML Information Set [p.85]*], i.e. they define the [local name], [namespace name], [children] and [attributes] properties of an element information item. Type systems based upon other data models are generally accommodated by extensions to WSDL; see **6. Language Extensibility** [p.77] . In the case where they define information equivalent to that of a XML Schema global element declaration, they can more simply be treated as if they were such a declaration.

The properties of the Definitions component are as follows:

- {interfaces} OPTIONAL. A set of Interface components.
- {bindings} OPTIONAL. A set of Binding components.
- {services} OPTIONAL. A set of Service components.
- {element declarations} OPTIONAL. A set of named element declarations, each one isomorphic to a global element declaration as defined by XML Schema.
- {type definitions} OPTIONAL. A set of named type definitions, each one isomorphic to a global type definition as defined by XML Schema.

The set of interfaces/binding/services/etc. available in the Definitions component include those that are defined within the component itself and those that are imported and/or included. Note that at the component model level, there is no distinction between directly defined components vs. imported/included components.

The components directly defined within a single Definitions component are said to belong to the same *target namespace*. The target namespace therefore groups a set of related component definitions and represents an unambiguous name for the intended semantics of the collection of components. The target namespace URI SHOULD point to a human or machine processable document that directly or indirectly defines the intended semantics of those components.

Note that it is RECOMMENDED that the value of the `targetNamespace` *attribute information item* SHOULD be a dereferencible URI and that it resolve to a WSDL document which provides service description information for that namespace.

If a service description is split into multiple documents (which may be combined as needed via **4.1 Including Descriptions** [p.73]), then the `targetNamespace` *attribute information item* SHOULD resolve to a master document which includes all the WSDL documents needed for that service description.

This approach enables the WSDL component designators' fragment identifiers to be properly resolvable.

Imported components have different target namespace values from the Definitions component that is importing them. Thus importing is the mechanism to use components from one namespace in another set of definitions.

Each WSDL or type system component MUST be uniquely identified by its qualified name. That is, if two distinct components of the same kind (Interface, Binding etc.) are in the same target namespace, then their QNames MUST be unique. However, different kinds of components (e.g., an Interface component and a Binding component) MAY have the same QName. Thus, QNames of components must be unique within the space of those components in a given target namespace.

In addition to WSDL components and type system components, additional extension components MAY be added via extensibility **6. Language Extensibility** [p.77] . Further, additional properties to WSDL and type system components MAY also be added via extensibility.

2.1.2 XML Representation of Definitions Component

```
<definitions
      targetNamespace="xs:anyURI" >
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</definitions>
```

WSDL definitions are represented in XML by one or more WSDL Information Sets (Infosets), that is one or more `definitions` *element information item*s. A WSDL Infoset contains representations for a collection of WSDL components which share a common target namespace. A WSDL Infoset which contains one or more `import` *element information item*s **4.2 Importing Descriptions** [p.75] corresponds to a collection with components drawn from multiple target namespaces.

The targetNamespace URI MUST be an absolute URI (see [*IETF RFC 2396 [p.84]*]).

The `definitions` *element information item* has the following Infoset properties:

- A [local name] of `definitions` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `targetNamespace` *attribute information item* as described below in **2.1.2.1 targetNamespace attribute information item** [p.13] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

- Zero or more *element information item*s amongst its [children], in order as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `include` *element information item*s (see **4.1 Including Descriptions** [p.73])
     - Zero or more `import` *element information item*s (see **4.2 Importing Descriptions** [p.75])
     - Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
  3. An OPTIONAL `types` *element information item* (see **3. Types** [p.68]).
  4. Zero or more *element information item*s from among the following, in any order:
     - `interface` *element information item*s (see **2.2.2 XML Representation of Interface Component** [p.16]).
     - `binding` *element information item*s (see **2.9.2 XML Representation of Binding Component** [p.48]).
     - `service` *element information item*s (see **2.13.2 XML Representation of Service Component** [p.60]).
     - Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.1.2.1 `targetNamespace` *attribute information item*

The `targetNamespace` *attribute information item* defines the namespace affiliation of top-level components defined in this `definitions` *element information item*. Interfaces, Bindings and Services are top-level components.

The `targetNamespace` *attribute information item* has the following Infoset properties:

- A [local name] of `targetNamespace`
- A [namespace name] which has no value

The type of the `targetNamespace` *attribute information item* is *xs:anyURI*.

2.1.3 Mapping Definitions' XML Representation to Component Properties

The mapping between the properties of the Definitions component (see **2.1.1 The Definitions Component** [p.11]) and the XML Representation of the `definitions` *element information item* (see **2.1.2 XML Representation of Definitions Component** [p.12]) is described in Table 2-1 [p.14] .

Table 2-1. Mapping between Definitions Component Properties and XML Representation

Property	Mapping
{interfaces}	The set of Interface components corresponding to all the `interface` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported Interface components (see **4. Modularizing WSDL descriptions** [p.73]).
{bindings}	The set of Binding components corresponding to all the `binding` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported Binding components (see **4. Modularizing WSDL descriptions** [p.73]).
{services}	The set of Service components corresponding to all the `service` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported Service components (see **4. Modularizing WSDL descriptions** [p.73]).
{element declarations}	The set of element declarations corresponding to all the element declarations defined as descendants of the `types` *element information item*, if any, plus any imported element declarations. At a minimum this will include all the global element declarations defined by XML Schema `element` *element information item*s. It MAY also include any declarations from some other type system which describes the [local name], [namespace name], [attributes] and [children] properties of an *element information item*.

2.2 Interface

2.2.1 The Interface Component

An Interface component describes sequences of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a sequence of input and output messages, and an interface is a set of operations. Thus, an interface defines the design of the application.

An interface can optionally extend one or more other interfaces. To avoid circular definitions, an interface MUST NOT appear as an element of the set of interfaces it extends, either directly or indirectly. An interface contains all the operations defined by the interfaces it extends, along with any operations it directly defines. In the process, operation components that are equivalent per **2.16 Equivalence of Components** [p.67] are treated as one. The interface extension mechanism behaves in a similar way for all other components that can be defined inside an interface, namely Interface Fault, Feature and Property components.

Interfaces are named constructs and can be referred to by QName (see **2.18 QName resolution** [p.68]). For instance, Binding components refer to interfaces in this way.

The properties of the Interface component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {target namespace} REQUIRED. A *wsdls:anyURI*, as defined in **2.15.4 anyURI Type** [p.66] .
- {extended interfaces} OPTIONAL. A set of Interface components which this interface extends. This set MUST be closed under the operation of adding the values of the {extended interfaces} properties of all its members.
- {faults} OPTIONAL. A set of Interface Fault components. This set MUST include the values of the {faults} properties of all the interface definitions that are listed under the {extended interfaces} property of the component.
- {operations} OPTIONAL. A set of Interface Operation components. This set MUST include the values of the {operations} properties of all the Interface components that are listed under the {extended interfaces} property of the component.
- {features} OPTIONAL. A set of Feature components. This set MUST include the values of the {features} properties of all the Interface components that are listed under the {extended interfaces} property of the component.
- {properties} OPTIONAL. A set of Property components. This set MUST include the values of the {properties} properties of all the Interface components that are listed under the {extended interfaces} property of the component.

For each Interface component in the {interfaces} property of a definitions container, the combination of {name} and {target namespace} properties MUST be unique.

Additionally, an Interface component MUST satisfy the Operation Name Mapping requirement, as defined below. This requirement is intended to ensure that a received message can be uniquely mapped to a corresponding wsdl:operation.

2.2.1.1 Operation Name Mapping Requirement

Consider all Interface Operation components specified in the {operations} property of an Interface component. Further, consider all Message Reference components specified in the {message references} properties of said Interface Operation components. Further, consider all said Message Reference components that have the same value for their {direction} property (i.e., either the token *in* or the token *out*). If the {message content model} property of any of these Message Reference components has a value of "#any", or if more than one of these Message Reference components has a value of "#none", or if the qualified names of the global element declarations specified by the values of the {element} properties of these Message Reference components are not unique when considered together, then either one of the following two conditions MUST apply:

1. the {features} property of the Interface component MUST contain a Feature component, having a {required} property with a value of *true*, that unambiguously identifies the mechanism that a message sender is required to support in order to enable the message recipient to unambiguously determine the name of the Interface Operation component that is intended to be associated with the received message; or

2. the *element information item* for the Interface component MUST contain an extension element (i.e., an element that is not in the http://www.w3.org/2004/08/wsdl namespace), having a wsdl:required *attribute information item* with a value of "true", that unambiguously identifies the mechanism that a message sender is required to support in order to enable the message recipient to unambiguously determine the name of the Interface Operation component that is intended to be associated with the received message.

2.2.2 XML Representation of Interface Component

```
<definitions>
  <interface
        name="xs:NCName"
        extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> | <property /> ]*
  </interface>
</definitions>
```

The XML representation for an Interface component is an *element information item* with the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.2.2.1 name attribute information item with interface [owner]** [p.17] .
  - An OPTIONAL `extends` *attribute information item* as described below in **2.2.2.2 extends attribute information item** [p.17] .
  - An OPTIONAL `styleDefault` *attribute information item* as described below in **2.2.2.3 styleDefault attribute information item** [p.17] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `fault` *element information item*s **2.3.2 XML Representation of Interface Fault Component** [p.20] .

- Zero or more `operation` *element information item*s **2.4.3 XML Representation of Interface Operation Component** [p.28] .
- Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41] .
- Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44] .
- Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.2.2.1 `name` *attribute information item* with `interface` [owner]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* forms the QName of the interface.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.2.2.2 `extends` *attribute information item*

The `extends` *attribute information item* lists the interfaces that this interface derives from.

The `extends` *attribute information item* has the following Infoset properties:

- A [local name] of `extends`
- A [namespace name] which has no value

The type of the `extends` *attribute information item* is a list of *xs:QName*.

2.2.2.3 `styleDefault` *attribute information item*

The `styleDefault` *attribute information item* indicates the default style used to construct the {element} properties of {message references} of all operations contained within the [owner] `interface` .

The `styleDefault` *attribute information item* has the following Infoset properties:

- A [local name] of `styleDefault.`

- A [namespace name] which has no value.

The type of the `styleDefault` *attribute information item* is *list of xs:anyURI*. Moreover, the value of the `styleDefault` *attribute information item*, if present, MUST contain absolute URIs (see [*IETF RFC 2396 [p.84]*]).

2.2.3 Mapping Interface's XML Representation to Component Properties

The mapping between the properties of the Interface component (see **2.2.1 The Interface Component** [p.14]) and the XML Representation of the `interface` *element information item* (see **2.2.2 XML Representation of Interface Component** [p.16]) is as described in Table 2-2 [p.18] .

Table 2-2. Mapping between Interface Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*
{extended interfaces}	The set of Interface components resolved to by the values in the `extends` *attribute information item* if any, plus the set of Interface components in the {extended interfaces} property of those interface definitions, if any.
{faults}	The set of Interface Fault components corresponding to the `fault` *element information item*s in [children], if any, plus the set of Interface Fault components in the {faults} property of the Interface components in {extended interfaces}, if any.
{operations}	The set of Interface Operation components corresponding to the `operation` *element information item*s in [children], if any, plus the set of Interface Operation components in the {operations} property of the Interface components in {extended interfaces}, if any.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any, plus the set of Feature components in the {features} property of the Interface components in {extended interfaces}, if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any, plus the set of Property components in the {properties} property of the Interface components in {extended interfaces}, if any.

Note that, per **2.2.1 The Interface Component** [p.14] , the Interface components in the {extended interfaces} property of a given Interface component MUST NOT contain that Interface component in any of their {extended interfaces} properties, that is to say, recursive extension of interfaces is disallowed.

2.3 Interface Fault

2.3.1 The Interface Fault Component

A fault is an event that occurs during the execution of a message exchange that disrupts the normal flow of messages.

A fault is typically raised when a party is unable to communicate an error condition inside the normal message flow, or a party wishes to terminate a message exchange. A fault message may be used to communicate out of band information such as the reason for the error, the origin of the fault, as well as other informal diagnostics such as a program stack trace.

An Interface Fault component describes a fault that MAY occur during invocation of an operation of the interface. The Interface Fault component declares an abstract fault by naming it and indicating the contents of the fault message. When and how the fault message flows is indicated by the Interface Operation component **2.4 Interface Operation** [p.22] .

The Interface Fault component provides a clear mechanism to name and describe the set of faults an interface may generate. This allows operations to easily identify the individual faults they may generate by name. This mechanism allows the ready identification of the same fault occurring across multiple operations and referenced in multiple bindings as well as reducing duplication of description for an individual fault.

Note that faults other than the ones described in the Interface component can also be generated at run-time, i.e. faults are an open set.

The properties of the Interface Fault component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {target namespace} REQUIRED. A *wsdls:anyURI*, as defined in **2.15.4 anyURI Type** [p.66] .
- {element} OPTIONAL. A reference to an XML element declaration in the {element declarations} property of **2.1.1 The Definitions Component** [p.11] . This element represents the content or "payload" of the fault.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

If a type system NOT based on the XML Infoset [*XML Information Set [p.85]*] is in use (as considered in **3.2 Using Other Schema Languages** [p.72]) then additional properties would need to be added to the Fault Component (along with extensibility attributes to its XML representation) to allow associating such message types with the message reference.

For each Interface Fault component in the {faults} property of an Interface component, the combination of {name} and {target namespace} properties must be unique.

Interface Fault components are local to Interface components; they cannot be referred to by QName, despite having both {name} and {target namespace} properties. That is, two Interface components sharing the same {target namespace} property but with different {name} properties MAY contain Interface Fault components which share the same {name} property. Thus, the {name} and {target namespace} properties of the Interface Fault components are not sufficient to form the unique identity of an Interface Fault component. To uniquely identify an Interface Fault component one must first identify the Interface component (by QName) and then identify the Interface Fault within that Interface component (by a further QName).

In cases where, due to an interface extending one or more other interfaces, two or more Interface Faults components have the same value for their {name} and {target namespace} properties, then the component models of those Interface Fault components MUST be equivalent (see **2.16 Equivalence of Components** [p.67]). If the Interface Fault components are equivalent then they are considered to collapse into a single component. It is an error if two Interface Fault components have the same value for their {name} and {target namespace} properties but are not equivalent.

Note that, due to the above rules, if two interfaces that have the same value for their {target namespace} property also have one or more faults that have the same value for their {name} property then those two interfaces cannot both form part of the derivation chain of a derived interface unless those faults are the same fault.

Note:

For the above reason, it is considered good practice to ensure, where necessary, that the {name} property of Interface Fault components within a namespace are unique, thus allowing such derivation to occur without inadvertent error.

2.3.2 XML Representation of Interface Fault Component

```
<definitions>
  <interface>
    <fault
          name="xs:NCName"
          element="xs:QName"? >
      <documentation />?
      [ <feature /> | <property /> ]*
    </fault>
  </interface>
</definitions>
```

The XML representation for an Interface Fault component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:

- A REQUIRED `name` *attribute information item* as described below in **2.3.2.1 name attribute information item with fault [owner]** [p.21] .
- An OPTIONAL `element` *attribute information item* as described below in **2.3.2.2 element attribute information item with fault [owner]** [p.21] .
- Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.3.2.1 `name` *attribute information item* with `fault` [owner]

The `name` *attribute information item* identifies a given `fault` *element information item* inside a given `interface` *element information item*.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.3.2.2 `element` *attribute information item* with `fault` [owner]

The `element` *attribute information item* refers, by QName, to an element declaration component.

The `element` *attribute information item* has the following Infoset properties:

- A [local name] of `element` .
- A [namespace name] which has no value.

The type of the `element` *attribute information item* is *xs:QName*.

2.3.3 Mapping Interface Fault's XML Representation to Component Properties

The mapping between the properties of the Interface Fault component (see **2.3.1 The Interface Fault Component** [p.19]) and the XML Representation of the `fault` *element information item* (see **2.3.2 XML Representation of Interface Fault Component** [p.20]) is as described in Table 2-3 [p.22] .

Table 2-3. Mapping between Interface Fault Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*.
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* of the [parent] `interface` *element information item*.
{element}	The element declaration from the {element declarations} property of **2.1.1 The Definitions Component** [p.11] resolved to by the value of the `element` *attribute information item* if present, otherwise empty. It is an error for the `element` *attribute information item* to have a value and for it to not resolve to a global element declaration from the {element declarations} property of **2.1.1 The Definitions Component** [p.11] .
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.4 Interface Operation

2.4.1 The Interface Operation Component

An Interface Operation component describes an operation that a given interface supports. An operation is an interaction with the service consisting of a set (ordinary and fault) messages exchanged between the service and the other roles involved in the interaction, in particular the service requester. The sequencing and cardinality of the messages involved in a particular interaction is governed by the *message exchange pattern* used by the operation (see {message exchange pattern} property).

A message exchange pattern defines placeholders for messages, the participants in the pattern (i.e., the sources and sinks of the messages), and the cardinality and sequencing of messages exchanged by the participants. The message placeholders are associated with specific message types by the operation that uses the pattern by means of message and fault references (see {message references} and {fault references} properties). The service whose operation is using the pattern becomes one of the participants of the pattern. This specification does not define a machine understandable language for defining message exchange patterns, nor does it define any specific patterns. The companion specification, [*WSDL 2.0 Predefined Extensions [p.85]*] defines a set of such patterns and defines identifying URIs any of which

MAY be used as the value of the {message exchange pattern} property.

The properties of the Interface Operation component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {target namespace} REQUIRED. A *wsdls:anyURI*, as defined in **2.15.4 anyURI Type** [p.66] .
- {message exchange pattern} REQUIRED. A *wsdls:anyURI* identifying the message exchange pattern used by the operation. This URI MUST be an absolute URI (see [*IETF RFC 2396 [p.84]*]).
- {message references} OPTIONAL. A set of Message Reference components for the ordinary messages the operation accepts or sends. (See **2.5 Message Reference** [p.31] .)
- {fault references} OPTIONAL. A set of Fault Reference components for the fault messages the operation accepts or sends. (See **2.6 Fault Reference** [p.35] .)
- {style} OPTIONAL. A set of *wsdls:anyURI*s identifying the rules that were used to construct the {element} properties of {message references}. (See **2.4.1.1 Operation Style** [p.24] .) These URIs MUST be absolute URIs (see [*IETF RFC 2396 [p.84]*]).
- {safety} REQUIRED. A *wsdls:boolean* (see **2.15.6 boolean Type** [p.66]) indicating whether the operation is asserted to be safe (as defined in Section 3.5 of [*Web Architecture [p.86]*]) for users of the described service to invoke. If this property is false, then no assertion has been made about the safety of the operation, thus the operation MAY or MAY NOT be safe. However, an operation SHOULD be marked safe if it meets the criteria for a safe interaction defined in Section 3.5 of [*Web Architecture [p.86]*].
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Interface Operation component in the {operations} property of an Interface component, the combination of {name} and {target namespace} properties MUST be unique.

Interface Operation components are local to Interface components; they cannot be referred to by QName, despite having both {name} and {target namespace} properties. That is, two Interface components sharing the same {target namespace} property but with different {name} properties MAY contain Interface Operation components which share the same {name} property. Thus, the {name} and {target namespace} properties of the Interface Operation components are not sufficient to uniquely identify an Interface Operation component. In order to uniquely identify an Interface Operation component, one must first identify the Interface component (by QName) and then identify the Interface Operation within that Interface component (by a further QName).

In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation components have the same value for their {name} and {target namespace} properties, then the component models of those Interface Operation components MUST be equivalent (see **2.16 Equivalence of Components** [p.67]). If the Interface Operation components are equivalent then they are considered to collapse into a single component. It is an error if two Interface Operation components have the same value

for their {name} and {target namespace} properties but are not equivalent.

Note that, due to the above rules, if two interfaces that have the same value for their {target namespace} property also have one or more operations that have the same value for their {name} property then those two interfaces cannot both form part of the derivation chain of a derived interface unless those operations are the same operation.

Note:

For the above reason, it is considered good practice to ensure, where necessary, that the {name} property of Interface Operation components within a namespace are unique, thus allowing such derivation to occur without inadvertent error.

2.4.1.1 Operation Style

If the {style} property of an Interface Operation component has a value then that value (a set of URIs) implies the rules that were used to define the {element} properties (or other property which defines the content of the message properties; see **3.2 Using Other Schema Languages** [p.72]) of *all* the Message Reference components which are members of the {message references} property of that component.

Note that the property MAY not have any value. If this property has a value (a set of URIs), then for each individual URI that is an element of that set, the rules implied by that URI (such as rules that govern the schemas) MUST be followed or it is an error. So, if the set of URIs has more than one item in it, then the rules implied by ALL the URIs must be adhered to by the content definitions.

This specification defines the following pre-defined operation style:

- RPC Style (see **2.4.2 RPC Style** [p.24])

2.4.2 RPC Style

The RPC style is selected by assigning to an Interface Operation component's {style} property the value *http://www.w3.org/2004/08/wsdl/style/rpc*.

The RPC style MUST NOT be used for Interface Operation components whose {message exchange pattern} property has a value other than 'http://www.w3.org/2004/08/wsdl/in-only' or 'http://www.w3.org/2004/08/wsdl/in-out'.

When this value is used, the associated messages MUST conform to the rules below, described using XML Schema [*XML Schema: Structures [p.85]*]. Note that operations containing messages described by other type systems may also indicate use of the RPC style, as long as they are constructed in such a way as to follow these rules.

If the Interface Operation component uses a {message exchange pattern} for which there is no output element, such as 'http://www.w3.org/2004/08/wsdl/in-only', then the conditions stated below that refer to output elements MUST be considered to be implicitely satisfied.

- The content model of input and output {element} elements MUST be defined using a complex type that contains a sequence from XML Schema.
- The sequence MUST only contain elements. It MUST NOT contain other structures such as xs:choice.
- The sequence MUST contain only local element children. Note that these child elements MAY contain the following attributes: nillable, minOccurs and maxOccurs.
- The LocalPart of input element's QName MUST be the same as the Interface operation component's name.
- The LocalPart of the output element's QName is obtained by concatenating the name of the operation and the string value "Response".
- Input and output elements MUST both be in the same namespace.
- The complex type that defines the body of an input or an output element MUST NOT contain any attributes.
- If elements with the same qualified name appear as children of both the input and output elements, then they MUST both be declared using the same type.
- The input or output sequence MUST NOT contain multiple children elements declared with the same name.

2.4.2.1 `wrpc:signature` Extension

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

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

- {rpc-signature} REQUIRED. A list of pairs *(q, t)* whose first component is of type *wsdls:QName* (as defined by **2.15.4 anyURI Type** [p.66]) and whose second component is of type *wsdls:Token* (as defined by **2.15.2 Token Type** [p.66]). Values for the second component MUST be chosen among the following four: "#in", "#out", "#inout" "#return".

The value of the {rpc-signature} property MUST satisfy the following conditions:

- The value of the first component of each pair *(q, t)* MUST be unique within the list.
- For each child element of the input and output messages of the operation, a pair *(q, t)* whose first component *q* is equal to the qualified name of that element MUST be present in the list, with the caveat that elements that appear with cardinality greater than one MUST be treated as as a single element.

- For each pair *(q, #in)*, there MUST be a child element of the input element with a name of *q* and there MUST NOT be a child element of the output element with the same name.
- For each pair *(q, #out)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.
- For each pair *(q, #inout)*, there MUST be a child element of the input element with a name of *q* and there MUST be a child element of the output element with the same name. Furthermore, those two elements MUST have the same type.
- For each pair *(q, #return)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.

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

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

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

2. Filter the elements of this list into two lists, the first one *(L1)* comprising pairs whose *t* component is one of *{#in, #out, #inout}*, the second *(L2)* pairs whose *t* component is *#return*.

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

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

   and

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

   respectively.

3. Then the formal signature of the function is

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

   i.e.

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

2.4.2.2 XML Representation of the `wrpc:signature` Extension

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

- A [local name] of `signature`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl/rpc"

The type of the `name` *attribute information item* is a list type whose item type is the union of the *xs:QName* type and the subtype of the *xs:Token* type restricted to the following four values: "#in", "#out", "#inout", "#return". See Example 2-1 [p.27] for a definition of this type.

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

Example 2-1. Definition of the wrpc:signature extension

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

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

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

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

2.4.2.3 `wrpc:signature` Extension Mapping To Properties of an Interface Operation Component

A `wrpc:signature` extension *attribute information item* is mapped to the following property of the Interface Operation component (see **2.4.1 The Interface Operation Component** [p.22]) defined by its [owner].

Table 2-4. Mapping of a `wrpc:signature` Extension to Interface Operation Component Properties

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

2.4.3 XML Representation of Interface Operation Component

```
<definitions>
  <interface>
    <operation
          name="xs:NCName"
          pattern="xs:anyURI"
          style="list of xs:anyURI"?
          safe="xs:boolean"? >
      <documentation />?
      [ <feature /> | <property /> |
        [ <input /> | <output /> | <infault /> | <outfault /> ]+
      ]*
    </operation>
  </interface>
</definitions>
```

The XML representation for an Interface Operation component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.4.3.1 name attribute information item with operation [owner]** [p.29] .
  - A REQUIRED `pattern` *attribute information item* as described below in **2.4.3.2 pattern attribute information item with operation [owner]** [p.30] .
  - An OPTIONAL `style` *attribute information item* as described below in **2.4.3.3 style attribute information item with operation [owner]** [p.30] .
  - An OPTIONAL `safe` *attribute information item* as described below in **2.4.3.4 safe attribute information item with operation [owner]** [p.30] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `input` *element information item*s (see **2.5.2 XML Representation of Message Reference Component** [p.32]).
     - Zero or more `output` *element information item*s (see **2.5.2 XML Representation of Message Reference Component** [p.32]).
     - Zero or more `infault` *element information item*s (see **2.6.2 XML Representation of Fault Reference Component** [p.36]).
     - Zero or more `outfault` *element information item*s (see **2.6.2 XML Representation of Fault Reference Component** [p.36]).
     - A `feature` *element information item* (see **2.7.2 XML Representation of Feature Component** [p.41]).
     - A `property` *element information item* (see **2.8.2 XML Representation of Property Component** [p.44]).
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- At least one of the [children] MUST be an `input` , `output` , `infault` , or `outfault` *element information item*.

2.4.3.1 `name` *attribute information item* with `operation` [owner]

The `name` *attribute information item* identifies a given `operation` *element information item* inside a given `interface` *element information item*.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.4.3.2 `pattern` *attribute information item* with `operation` [owner]

The `pattern` *attribute information item* identifies the message exchange pattern a given operation uses.

The `pattern` *attribute information item* has the following Infoset properties:

- A [local name] of `pattern`
- A [namespace name] which has no value

The type of the `pattern` *attribute information item* is *xs:anyURI*.

2.4.3.3 `style` *attribute information item* with `operation` [owner]

The `style` *attribute information item* indicates the rules that were used to construct the {element} properties of the Message Reference components which are members of the {message references} property of the [owner] operation.

The `style` *attribute information item* has the following Infoset properties:

- A [local name] of `style`
- A [namespace name] which has no value

The type of the `style` *attribute information item* is *list of xs:anyURI*.

2.4.3.4 `safe` *attribute information item* with `operation` [owner]

The `safe` *attribute information item* indicates whether the operation is *safe* or not.

The `safe` *attribute information item* has the following Infoset properties:

- A [local name] of `safe`
- A [namespace name] which has no value

The type of the `safe` *attribute information item* is *xs:boolean* and does not have a default value.

2.4.4 Mapping Interface Operation's XML Representation to Component Properties

The mapping between the properties of the Interface Operation component (see **2.4.1 The Interface Operation Component** [p.22]) and the XML Representation of the `operation` *element information item* (see **2.4.3 XML Representation of Interface Operation Component** [p.28]) is as described in Table 2-5 [p.30] .

Table 2-5. Mapping between Interface Operation Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* of the [parent] `interface` *element information item*.
{message exchange pattern}	The actual value of the `pattern` *attribute information item*
{message references}	The set of message references corresponding to the `input` and `output` *element information item*s in [children], if any.
{fault references}	The set of fault references corresponding to the `infault` and `outfault` *element information item*s in [children], if any.
{style}	The set containing the URIs in the actual value of the `style` *attribute information item* if present, otherwise the set containing the URIs in the actual value of the `styleDefault` *attribute information item* of the [parent] `interface` *element information item* if present, otherwise empty.
{safety}	The actual value of the `safe` *attribute information item* if present, otherwise the value *false*.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.5 Message Reference

2.5.1 The Message Reference Component

A Message Reference component associates a defined type with a message exchanged in an operation. By default, the type system is based upon the XML Infoset [*XML Information Set [p.85]*].

A message exchange pattern defines a set of placeholder messages that participate in the pattern and assigns them unique message labels within the pattern (e.g. 'In', 'Out'). The purpose of a Message Reference component is to associate an actual message type (XML element declaration or some other declaration (see **3.2 Using Other Schema Languages** [p.72]) for message content) with a message in the pattern, as identified by its message label. Later, when the message exchange pattern is instantiated, messages corresponding to that particular label will follow the type assignment made by the Message Reference component.

The properties of the Message Reference component are as follows:

- {message label} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] . This property identifies the role this message plays in the {message exchange pattern} of the Interface Operation component this is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.

- {direction} REQUIRED. A *wsdls:Token* with one of the values *in* or *out*, indicating whether the message is coming to the service or going from the service, respectively. The direction MUST be the same as the direction of the message identified by the {message label} property in the {message exchange pattern} of the Interface Operation component this is contained within.

- {message content model} OPTIONAL. A *wsdls:token* with one of the values *#any*, *#none*, or *#element*. A value of *#any* indicates that the message content is any single element. A value of *#none* indicates there is no message content. A value of *#element* indicates that the message consists of a single element described by the global element declaration reference by the {element} property.

- {element} OPTIONAL. A reference to an XML element declaration in the {element declarations} property of **2.1.1 The Definitions Component** [p.11] . This element represents the content or "payload" of the message. When the {message content model} property has the value *#any* or *#none* the {element} property MUST be empty.

- {features} OPTIONAL. A set of Feature components.

- {properties} OPTIONAL. A set of Property components.

If a type system not based upon the XML Infoset is in use (as considered in **3.2 Using Other Schema Languages** [p.72]) then additional properties would need to be added to the Message Reference Component (along with extensibility attributes to its XML representation) to allow associating such message types with the message reference.

For each Message Reference component in the {message references} property of an Interface Operation component, its {message label} property MUST be unique.

2.5.2 XML Representation of Message Reference Component

```
<definitions>
  <interface>
    <operation>
      <input
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:Token"? >
        <documentation />?
        [ <feature /> | <property /> ]*
      </input>
      <output
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:Token"? >
        <documentation />?
        [ <feature /> | <property /> ]*
```

```
      </output>
    </operation>
  </interface>
</definitions>
```

The XML representation for a Message Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `input` or `output`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- Zero or more *attribute information item*s amongst its [attributes] as follows:
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.5.2.1 messageLabel attribute information item with input, or output [owner]** [p.33] .

    If the {message exchange pattern} of the Interface Operation component has only one message with a given value for {direction}, then the `messageLabel` *attribute information item* is optional for the XML representation of the Message Reference component with that {direction}.
  - An OPTIONAL `element` *attribute information item* as described below in **2.5.2.2 element attribute information item with input, or output [owner]** [p.34] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.5.2.1 `messageLabel` *attribute information item* with `input` , or `output` [owner]

The `messageLabel` *attribute information item* identifies the role of this message in the message exchange pattern of the given `operation` *element information item*.

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel`
- A [namespace name] which has no value

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.5.2.2 `element` *attribute information item* with `input` , or `output` [owner]

The `element` *attribute information item* has the following Infoset properties:

- A [local name] of `element` .
- A [namespace name] which has no value.

The type of the `element` *attribute information item* is a union of *xs:QName* and *xs:Token* where the allowed token values are *#any* or *#none*.

2.5.3 Mapping Message Reference's XML Representation to Component Properties

The mapping between the properties of the Message Reference component (see **2.5.1 The Message Reference Component** [p.31]) and the XML Representation of the message reference *element information item* (see **2.5.2 XML Representation of Message Reference Component** [p.32]) is as described in Table 2-6 [p.34] .

Table 2-6. Mapping between Message Reference Component Properties and XML Representation

Property	Mapping
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with same {direction} from the {message exchange pattern} of the Interface Operation component, provided there is exactly one such message; otherwise it is an error.
{direction}	If the [local name] of the *element information item* is `input` then "in", else if the [local name] of the *element information item* is `output` then "out".
{message content model}	If the `element` *attribute information item* is present and its value is a QName, then *#element*. Otherwise the actual value of the `element` *attribute information item*, if any, otherwise empty.
{element}	If the `element` *attribute information item* is present and its value is a QName, then the element declaration from the {element declarations} property of **2.1.1 The Definitions Component** [p.11] resolved to by the value of the `element` *attribute information item*, otherwise empty. It is an error for the `element` *attribute information item* to have a value and for it to not resolve to a global element declaration from the {element declarations} property of **2.1.1 The Definitions Component** [p.11] .
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.6 Fault Reference

2.6.1 The Fault Reference Component

A Fault Reference component associates a defined type, specified by an Interface Fault component, to a fault message exchanged in an operation.

A message exchange pattern defines a set of placeholder messages that participate in the pattern and assigns them unique message labels within the pattern (e.g. 'In', 'Out'). The purpose of a Fault Reference component is to associate an actual message type (XML element declaration or some other declaration (see **3.2 Using Other Schema Languages** [p.72]) for message content, as specified by an Interface Fault component) with a fault message occurring in the pattern. In order to identify the fault message it describes, the Fault Reference component uses the message label of the message the fault is associated with as a key.

The companion specification [*WSDL 2.0 Predefined Extensions [p.85]*] defines two *fault patterns* that a given message exchange pattern may use. For the pattern *fault-replaces-message*, the message that the fault relates to identifies the message *in place of which* the declared fault message will occur. Thus, the fault message will travel in the *same* direction as the message it replaces in the pattern. For the pattern *message-triggers-fault*, the message that the fault relates to identifies the message after which the

indicated fault may occur, in the opposite direction of the referred to message. That is, the fault message will travel in the *opposite* direction of the message it comes after in the pattern.

More than one Fault Reference component may refer to the same message label. This allows one to indicate that there is more than one type of fault that is related to that message.

The properties of the Fault Reference component are as follows:

- {fault reference} REQUIRED. An Interface Fault component in the {faults} property of the parent Interface Operation component's parent Interface component. Identifying the Interface Fault component therefore indirectly defines the actual content or payload of the fault message.
- {message label} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] . This property identifies the message this fault relates to among those defined in the {message exchange pattern} property of the Interface Operation component it is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
- {direction} REQUIRED. A *wsdls:Token* with one of the values *in* or *out*, indicating whether the fault is coming to the service or going from the service, respectively. The direction MUST be consistent with the direction implied by the fault rule used in the message exchange pattern of the operation. For example, if the fault rule *fault-replaces-message* is used, then a fault which refers to an outgoing message would have a {direction} property value of *out*. On the other hand, if the fault rule *message-triggers-fault* is used, then a fault which refers to an outgoing message would have a {direction} property value of *in* as the fault travels in the opposite direction of the message.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

2.6.2 XML Representation of Fault Reference Component

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

The XML representation for a Fault Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `infault` or `outfault`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.6.2.1 ref attribute information item with infault, or outfault [owner]** [p.37] .
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.6.2.2 messageLabel attribute information item with infault, or outfault [owner]** [p.38] .

    If the {message exchange pattern} of the Interface Operation component has only one message with a given value for {direction}, the `messageLabel` *attribute information item* is optional for the XML representation of any Fault Reference component with the same value for {direction} (if the *fault pattern* of the {message exchange pattern} is *fault-replaces-message*) or of any Fault Reference component with the opposite value for {direction} (if the *fault pattern* is *message-triggers-fault*).
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.6.2.1 `ref` *attribute information item* with `infault` , or `outfault` [owner]

The `ref` *attribute information item* refers to a fault component.

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`
- A [namespace name] which has no value

The type of the `fault` *attribute information item* is *xs:QName*.

2.6.2.2 `messageLabel` *attribute information item* with `infault` , or `outfault` [owner]

The `messageLabel` *attribute information item* identifies the message in the message exchange pattern of the given `operation` *element information item* to which this fault is related to.

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel`
- A [namespace name] which has no value

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.6.3 Mapping Fault Reference's XML Representation to Component Properties

The mapping between the properties of the Fault Reference component (see **2.6.1 The Fault Reference Component** [p.35]) and the XML Representation of the message reference *element information item* (see **2.6.2 XML Representation of Fault Reference Component** [p.36]) is as described in Table 2-7 [p.38] .

Table 2-7. Mapping between Fault Reference Component Properties and XML Representation

Property	Mapping
{fault reference}	The actual value of the `ref` *attribute information item*
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with the same {direction} from the {message exchange pattern} of the Interface Operation component, provided there is exactly one such message and the *fault pattern* of the {message exchange pattern} is *fault-replaces-message*; otherwise the {message reference} property of the message with the opposite {direction}, provided there is exactly one such message and the *fault pattern* is *message-triggers-fault*; otherwise it is an error.
{direction}	If the [local name] of the *element information item* is `infault` then "in", else if the [local name] of the *element information item* is `outfault` then "out".
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.7 Feature

2.7.1 The Feature Component

A feature component describes an abstract piece of functionality typically associated with the exchange of messages between communicating parties. Although WSDL poses no constraints on the potential scope of such features, examples might include "reliability", "security", "correlation", and "routing". The presence of a feature component in a WSDL description indicates that the service supports the feature and may require a requester agent that interacts with the service to use that feature. Each Feature is identified by its URI.

The properties of the Feature component are as follows:

- {name} REQUIRED. A *wsdls:anyURI* as defined in **2.15.4 anyURI Type** [p.66] . This URI MUST be absolute as defined by [*IETF RFC 2396 [p.84]*]. This URI SHOULD be dereferenceable to a document that directly or indirectly defines the meaning and use of the Feature that it identifies.

- {required} REQUIRED. A *wsdls:boolean* value as defined by **2.15.6 boolean Type** [p.66] . If the value of this property is *true*, then the requester agent MUST use the Feature that is identified by the {name} URI. Otherwise, the requester agent MAY use the Feature that is identified by the {name} URI. In either case, if the requester agent does use the Feature that is identified by the {name} URI, then the requester agent MUST obey all semantics implied by the definition of that Feature.

2.7.1.1 Feature Composition Model

The set of features which are required or available for a given component consists of the combined set of ALL feature declarations applicable to that component. A feature is applicable to a component if:

- it is asserted directly within that component, or

- it is asserted in a containing component, or

- it is asserted in a component referred to by the current component.

If a given feature is asserted at multiple locations, then the value of that feature at a particular component is that given by the nearest assertion in lexical scoping order. Following these rules, the set of features applicable at each component are as follows:

- Interface component: all features asserted within the interface component.

- Interface Fault component: all features asserted within the interface fault component and those within the parent interface component.

- Interface Operation component: all features asserted within the interface operation component and those within the parent interface component.

- Message Reference component: all features asserted within the message reference component, those within the parent interface operation component and those within its parent interface component.
- Binding component: all features asserted within the binding component and those within the interface component referred to by the binding component (if any).
- Binding Fault component: all features asserted within the binding fault component, those within the parent binding component and those within the interface component referred to by the binding component (if any).
- Binding Operation component: all features asserted within the binding operation component, those within the parent binding component and those within the interface component referred to by the binding component (if any).
- Binding Message Reference component: all features asserted within the binding message reference component, those within the parent binding operation component, those within its parent binding component and those within the interface component referred to by the binding component (if any).

2.7.1.1.1 Example of Feature Composition Model

In the following example, the `depositFunds` operation on the `BankService` has to be used with the `ISO9001` , the `notarization` and the `secure-channel` features; they are all in scope. The fact that the `notarization` feature is declared both in the operation and in the binding has no effect.

```
<definitions targetNamespace="http://example.com/bank"
     xmlns:ns1="http://example.com/bank">
  <interface name="ns1:Bank">
    <!-- All implementations of this interface must be secure -->
    <feature uri="http://example.com/secure-channel"
             required="true"/>
    <operation name="withdrawFunds">
      <!-- This operation must have ACID properties -->
      <feature uri="http://example.com/transaction"
               required="true"/>
      ...
    </operation>
    <operation name="depositFunds">
      <!-- This operation requires notarization -->
      <feature uri="http://example.com/notarization"
               required="true"/>
      ...
    </operation>
  </interface>

  <binding name="ns1:BankSOAPBinding">
    <!-- This particular binding requires ISO9001
         compliance to be verifiable -->
    <feature uri="http://example.com/ISO9001"
             required="true"/>
    <!-- This binding also requires notarization -->
    <feature uri="http://example.com/notarization"
             required="true"/>
  </binding>
```

```
  <service name="ns1:BankService"
           interface="tns:Bank">
    <endpoint binding="ns1:BankSOAPBinding">
      ...
    </endpoint>
  </service>
</definitions>
```

2.7.2 XML Representation of Feature Component

```
<feature
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
</feature>
```

The XML representation for a Feature component is an *element information item* with the following Infoset properties:

- A [local name] of `feature`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `uri` *attribute information item* as described below in **2.7.2.1 uri attribute information item with feature [owner]** [p.41] .
  - An OPTIONAL `required` *attribute information item* as described below in **2.7.2.2 required attribute information item with feature [owner]** [p.42] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item*s amongst its [children], in order as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.7.2.1 `uri` *attribute information item* with `feature` [owner]

The `uri` *attribute information item* specifies the URI of the feature.

The `uri` *attribute information item* has the following Infoset properties:

- A [local name] of `uri`

- A [namespace name] which has no value

The type of the `uri` *attribute information item* is `xs:anyURI` .

2.7.2.2 `required` *attribute information item* with `feature` [owner]

The `required` *attribute information item* specifies whether the use of the feature is mandatory or optional.

The `required` *attribute information item* has the following Infoset properties:

- A [local name] of `required`
- A [namespace name] which has no value

The type of the `required` *attribute information item* is `xs:boolean` .

2.7.3 Mapping Feature's XML Representation to Component Properties

The mapping between the properties of the Feature component (see **2.7.1 The Feature Component** [p.39]) and the XML Representation of the `feature` *element information item* (see **2.7.2 XML Representation of Feature Component** [p.41]) is as described in Table 2-8 [p.42] .

Table 2-8. Mapping between Feature Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `uri` *attribute information item*
{required}	The actual value of the `required` *attribute information item* if present, otherwise "false".

2.8 Property

2.8.1 The Property Component

A "property" in the Features and Properties architecture represents a named runtime value which affects the behaviour of some aspect of a Web service interaction, much like an environment variable. For example, a reliable messaging SOAP module may specify a property to control the number of retries in the case of network failure. WSDL documents may specify the value constraints for these properties by referring to a Schema type, or by specifying a particular value. Properties, and hence property values, can be shared amongst features/bindings/modules, and are named with URIs precisely to allow this type of sharing.

The properties of the Property component are as follows:

- {name} REQUIRED. A *wsdls:anyURI* as defined in **2.15.4 anyURI Type** [p.66] . This URI MUST be absolute as defined by [*IETF RFC 2396 [p.84]*]. This URI SHOULD be dereferenceable to a document that directly or indirectly defines the meaning and use of the Property that it identifies.

- {required} REQUIRED. A *wsdls:boolean* value as defined by **2.15.6 boolean Type** [p.66] . If the {required} property is *true*, then the requester agent MUST use the Property that is identified by the {name} URI. Otherwise, the requester agent MAY use the Property that is identified by the {name} URI. In either case, if the requester agent does use the Property that is identified by the {name} URI, then the requester agent MUST obey all semantics implied by the definition of that Property.

- {value constraint} OPTIONAL. A type definition constraining the value of the property, or the token *#value* if the {value} property is not empty.

- {value} OPTIONAL. The value of the property, an ordered list of child information items, as specified by the [children] property of *element information item*s in [*XML Information Set [p.85]*].

If a type system not based upon the XML Infoset is in use (as considered in **3.2 Using Other Schema Languages** [p.72]) then additional properties would need to be added to the Property Component (along with extensibility attributes to its XML representation) to allow using such a type system to describe values and constraints for properties.

2.8.1.1 Property Composition Model

At runtime, the behavior of features, (SOAP) modules and bindings may be affected by the values of in-scope properties. Properties combine into a virtual "execution context" which maps property names (URIs) to constraints. Each property URI MAY therefore be associated with AT MOST one property constraint for a given interaction.

The set of properties which are required or available for a given component consists of the combined set of ALL property declarations applicable to that componment. A property is applicable to a component if:

- it is asserted directly within that component, or

- it is asserted in a containing component, or

- it is asserted in a component referred to by the current component.

If a given property is asserted at multiple locations, then the value of that property at a particular component is that given by the nearest assertion in lexical scoping order. Following these rules, the set of properties applicable at each component are as follows:

- Interface component: all properties asserted within the interface component.

- Interface Fault component: all properties asserted within the interface fault component and those within the parent interface component.

- Interface Operation component: all properties asserted within the interface operation component and those within the parent interface component.

- Message Reference component: all properties asserted within the message reference component, those within the parent interface operation component and those within its parent interface component.

- Binding component: all properties asserted within the binding component and those within the interface component referred to by the binding component (if any).
- Binding Fault component: all properties asserted within the binding fault component, those within the parent binding component and those within the interface component referred to by the binding component (if any).
- Binding Operation component: all properties asserted within the binding operation component, those within the parent binding component and those within the interface component referred to by the binding component (if any).
- Binding Message Reference component: all properties asserted within the binding message reference component, those within the parent binding operation component, those within its parent binding component and those within the interface component referred to by the binding component (if any).

Note that, in the text above, "property constraint" (or, simply, "constraint") is used to mean EITHER a `constraint` inside a property component OR a `value` , since `value` may be considered a special case of `constraint` .

2.8.2 XML Representation of Property Component

```
<property
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
  [ <value /> | <constraint /> ]?
</property>
```

The XML representation for a Property component is an *element information item* with the following Infoset properties:

- A [local name] of `property`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `uri` *attribute information item* as described below in **2.8.2.1 uri attribute information item with property [owner]** [p.45] .
  - An OPTIONAL `required` *attribute information item* as described below in **2.8.2.2 required attribute information item with property [owner]** [p.45] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- One or more *element information item*s amongst its [children], in order as follows:

1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
2. One OPTIONAL *element information item* from among the following:
   - A `value` *element information item* as described in **2.8.2.3 value element information item with property [parent]** [p.45]
   - A `constraint` *element information item* as described in **2.8.2.4 constraint element information item with property [parent]** [p.46]
3. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

2.8.2.1 `uri` *attribute information item* with `property` [owner]

The `uri` *attribute information item* specifies the URI of the property. It has the following Infoset properties:

- A [local name] of `uri`
- A [namespace name] which has no value

The type of the `uri` *attribute information item* is `xs:anyURI` .

2.8.2.2 `required` *attribute information item* with `property` [owner]

The `required` *attribute information item* specifies whether use of the property is mandatory or optional.

The `required` *attribute information item* has the following Infoset properties:

- A [local name] of `required`
- A [namespace name] which has no value

The type of the `required` *attribute information item* is `xs:boolean` .

2.8.2.3 `value` *element information item* with `property` [parent]

```
<property>
  <value>
    xs:anyType
  </value>
</property>
```

The `value` *element information item* specifies the value of the property. It has the following Infoset properties:

- A [local name] of `value`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"

The type of the `value` *element information item* is `xs:anyType` .

2.8.2.4 `constraint` *element information item* with `property` [parent]

```
<property>
  <constraint>
    xs:QName
  </constraint>
</property>
```

The `constraint` *element information item* specifies a constraint on the value of the property. It has the following Infoset properties:

- A [local name] of `constraint`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"

The type of the `constraint` *attribute information item* is `xs:QName` .

2.8.3 Mapping Property's XML Representation to Component Properties

The mapping between the properties of the Property component (see **2.8.1 The Property Component** [p.42]) and the XML Representation of the `property` *element information item* (see **2.8.2 XML Representation of Property Component** [p.44]) is as described in Table 2-9 [p.46] .

Table 2-9. Mapping between Property Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `uri` *attribute information item*.
{required}	The actual value of the `required` *attribute information item* if present, otherwise "false".
{value constraint}	If the `constraint` *element information item* is present, the type definition referred to by the value of this *element information item*. Otherwise, if the `value` *element information item* is present, the token *#value*, otherwise empty.
{value}	The value of the [children] property of the `value` *element information item*, if that element is present, otherwise empty.

2.9 Binding

2.9.1 The Binding Component

A Binding component describes a concrete message format and transmission protocol which may be used to define an endpoint (see **2.14 Endpoint** [p.62]). That is, a Binding component defines the implementation details necessary to accessing the service.

Binding components can be used to describe such information in a re-usable manner for any interface or specifically for a given interface. Furthermore, binding information MAY be specified on a per-operation basis (see **2.11.1 The Binding Operation Component** [p.53]) within an interface in addition to across all operations of an interface.

If a Binding component specifies any operation-specific binding details (by including Binding Operation components) or any fault binding details (by including Binding Fault components) then it MUST specify an interface the Binding component applies to, so as to indicate which interface the operations come from.

Conversely, a Binding component which omits any operation-specific binding details and any fault binding details MAY omit specifying an interface. Binding components that do not specify an interface MAY be used to specify operation-independent binding details for Service components with different interfaces. That is, such Binding components are reusable across one or more interfaces.

No concrete binding details are given in this specification. The companion specification, *WSDL (Version 2.0): Bindings* [*WSDL 2.0 Bindings [p.85]*] defines such bindings for SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework [p.86]*] and HTTP [*IETF RFC 2616 [p.86]*]. Other specifications MAY define additional binding details. Such specifications are expected to annotate the Binding component (and its sub-components) with additional properties and specify the mapping between those properties and the XML representation.

A Binding component which defines bindings for an Interface component MUST define bindings for all the operations of that Interface component. The bindings may occur via defaulting rules which allow one to specify default bindings for all operations (see, for example [*WSDL 2.0 Bindings [p.85]*]) or by directly listing each Operation component of the Interface component and defining bindings for them. Thus, it is an error for a Binding component to not define bindings for all the Operation components of the Interface component for which the Binding component purportedly defines bindings for.

Bindings are named constructs and can be referred to by QName (see **2.18 QName resolution** [p.68]). For instance, Endpoint components refer to bindings in this way.

The properties of the Binding component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {target namespace} REQUIRED. A *wsdls:anyURI* as defined in **2.15.4 anyURI Type** [p.66] .
- {interface} OPTIONAL. An Interface component indicating the interface for which binding information is being specified.

- {type} REQUIRED. A *wsdls:anyURI* as defined by **2.15.4 anyURI Type** [p.66] . This URI MUST be absolute as defined by [*IETF RFC 2396 [p.84]*]. The value of this URI indicates what kind of concrete binding details are contained within this Binding component. Specifications (such as [*WSDL 2.0 Bindings [p.85]*]) that define such concrete binding details MUST specify appropriate values for this property. The value of this property MAY be the namespace name of the extension elements or attributes which define those concrete binding details.
- {faults} OPTIONAL. A set of Binding Fault components.
- {operations} OPTIONAL. A set of Binding Operation components.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Binding component in the {bindings} property of a definitions container, the combination of {name} and {target namespace} properties must be unique.

2.9.2 XML Representation of Binding Component

```
<definitions>
  <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI" >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> | <property /> ]*
  </binding>
</definitions>
```

The XML representation for a Binding component is an *element information item* with the following Infoset properties:

- A [local name] of `binding`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.9.2.1 name attribute information item with binding [owner]** [p.49] .
  - An OPTIONAL `interface` *attribute information item* as described below in **2.9.2.2 interface attribute information item with binding [owner]** [p.49] .
  - An REQUIRED `type` *attribute information item* as described below in **2.9.2.3 type attribute information item with binding [owner]** [p.50] .

    - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

- Zero or more *element information item*s amongst its [children], in order, as follows:

    1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).

    2. Zero or more *element information item*s from among the following, in any order:

        - Zero or more `fault` *element information item*s (see **2.10.2 XML Representation of Binding Fault Component** [p.52]).

        - Zero or more `operation` *element information item*s (see **2.11.2 XML Representation of Binding Operation Component** [p.54]).

        - Zero or more `feature` *element information item*s (see **2.7.2 XML Representation of Feature Component** [p.41]).

        - Zero or more `property` *element information item*s (see **2.8.2 XML Representation of Property Component** [p.44]).

        - Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl". Such *element information item*s are considered to be binding extension elements(see **2.9.2.4 Binding extension elements** [p.50]).

2.9.2.1 `name` *attribute information item* with `binding` [owner]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the binding.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`

- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.9.2.2 `interface` *attribute information item* with `binding` [owner]

The `interface` *attribute information item* refers, by QName, to an Interface component.

The `interface` *attribute information item* has the following Infoset properties:

- A [local name] of `interface`

- A [namespace name] which has no value

The type of the `interface` *attribute information item* is *xs:QName*.

2.9.2.3 `type` *attribute information item* with `binding` [owner]

The `type` *attribute information item* identifies the kind of binding details contained in the Binding component.

The `type` *attribute information item* has the following Infoset properties:

- A [local name] of `type`
- A [namespace name] which has no value

The type of the `type` *attribute information item* is *xs:anyURI*.

2.9.2.4 Binding extension elements

Binding extension elements are used to provide information specific to a particular binding. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding component with additional properties and specify the mapping between those properties and the XML representation.

2.9.3 Mapping Binding's XML Representation to Component Properties

The mapping between the properties of the Binding component (see **2.9.1 The Binding Component** [p.47]) and the XML Representation of the `binding` *element information item* (see **2.9.2 XML Representation of Binding Component** [p.48]) is as described in Table 2-10 [p.50] .

Table 2-10. Mapping between Binding Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*.
{interface}	The Interface component resolved to by the actual value of the `interface` *attribute information item*, if any.
{type}	The actual value of the `type` *attribute information item*.
{faults}	The set of Binding Fault components corresponding to the `fault` *element information item*s in [children], if any.
{operations}	The set of Binding Operation components corresponding to the `operation` *element information item*s in [children], if any.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.10 Binding Fault

2.10.1 The Binding Fault Component

A Binding Fault component describes a concrete binding of a particular fault within an interface to a particular concrete message format. A particular fault of an interface is uniquely identified by the target namespace of the interface and the name of the fault within that interface.

Note that the fault does not occur by itself - it occurs as part of a message exchange as defined by an Interface Operation component (and its binding counterpart the Binding Operation component). Thus, the fault binding information specified in a Binding Fault component describes how faults that occur within a message exchange of an operation will be formatted.

The properties of the Binding Fault component are as follows:

- {fault reference} REQUIRED. An Interface Fault component in the {faults} property of the Interface component identified by the {interface} property of the parent Binding component. This is the Interface Fault component for which binding information is being specified.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Binding Fault component in the {faults} property of a Binding component, the {fault reference} property MUST be unique. That is, one cannot define multiple bindings for the same fault within a given Binding component.

2.10.2 XML Representation of Binding Fault Component

```
<definitions>
  <binding>
    <fault
          ref="xs:QName" >
      <documentation />?
      [ <feature /> | <property /> ]*
    </fault>
  </binding>
</definitions>
```

The XML representation for a Binding Fault component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.10.2.1 ref attribute information item with fault [owner]** [p.53] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl". Such *element information item*s are considered to be binding fault extension elements as described below (see **2.10.2.2 Binding Fault extension elements** [p.53]).

2.10.2.1 `ref` *attribute information item* with `fault` [owner]

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`
- A [namespace name] which has no value

The type of the `ref` *attribute information item* is *xs:QName*.

2.10.2.2 Binding Fault extension elements

Binding Fault extension elements are used to provide information specific to a particular fault in a binding. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Fault component with additional properties and specify the mapping between those properties and the XML representation.

2.10.3 Mapping Binding Fault's XML Representation to Component Properties

The mapping between the properties of the Binding Fault component (see **2.10.1 The Binding Fault Component** [p.51]) and the XML Representation of the `fault` *element information item* (see **2.10.2 XML Representation of Binding Fault Component** [p.52]) is as described in Table 2-11 [p.53] .

Table 2-11. Mapping between Binding Fault Component Properties and XML Representation

Property	Mapping
{fault reference}	The actual value of the `ref` *attribute information item*.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.11 Binding Operation

2.11.1 The Binding Operation Component

The Binding Operation component describes the concrete message format(s) and protocol interaction(s) associated with a particular interface operation for a given endpoint. A particular operation of an interface is uniquely identified by the target namespace of the interface and the name of the operation within that interface.

The properties of the Binding Operation component are as follows:

- {operation reference} REQUIRED. An Interface Operation component in the {operations} property of the Interface component identified by the {interface} property of the parent Binding component. This is the Interface Operation component for which binding information is being specified.
- {message references} OPTIONAL. A set of Binding Message Reference components
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Binding Operation component in the {operations} property of a Binding component, the {operation reference} property MUST be unique. That is, one cannot define multiple bindings for the same operation within a given Binding component.

2.11.2 XML Representation of Binding Operation Component

```
<definitions>
  <binding>
    <operation
          ref="xs:QName" >
      <documentation />?
      [ <input /> | <output /> | <feature /> | <property /> ]*
    </operation>
  </binding>
</definitions>
```

The XML representation for a Binding Operation component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.11.2.1 ref attribute information item with operation [owner]** [p.55] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `input` *element information item*s (see **2.12 Binding Message Reference** [p.56])

- Zero or more `output` *element information item*s (see **2.12 Binding Message Reference** [p.56])
- Zero or more `feature` *element information item*s (see **2.7.2 XML Representation of Feature Component** [p.41])
- Zero or more `property` *element information item*s (see **2.7.2 XML Representation of Feature Component** [p.41])
- Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl". Such *element information item*s are considered to be binding operation extension elements as described below (see **2.11.2.2 Binding Operation extension elements** [p.55]).

2.11.2.1 `ref` *attribute information item* with `operation` [owner]

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`
- A [namespace name] which has no value

The type of the `ref` *attribute information item* is *xs:QName*.

2.11.2.2 Binding Operation extension elements

Binding Operation extension elements are used to provide information specific to a particular operation in a binding. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Operation component with additional properties and specify the mapping between those properties and the XML representation.

2.11.3 Mapping Binding Operation's XML Representation to Component Properties

The mapping between the properties of the Binding Operation component (see **2.11.1 The Binding Operation Component** [p.53]) and the XML Representation of the `operation` *element information item* (see **2.11.2 XML Representation of Binding Operation Component** [p.54]) is as described in Table 2-12 [p.55] .

Table 2-12. Mapping between Binding Operation Component Properties and XML Representation

Property	Mapping
{operation reference}	The actual value of the `ref` *attribute information item*.
{messages references}	The set of Binding Message Reference components corresponding to the `input` and `output` *element information item*s in [children], if any.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.12 Binding Message Reference

2.12.1 The Binding Message Reference Component

A Binding Message Reference component describes a concrete binding of a particular message participating in an operation to a particular concrete message format.

The properties of the Binding Message Reference component are as follows:

- {message label} OPTIONAL. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] . The value of this property identifies the role that the message for which binding details are being specified plays in the {message exchange pattern} of the Interface Operation component being bound by the containing Binding Operation component.
- {direction} REQUIRED. A *wsdls:Token* with one of the values *in* or *out* indicating whether the message is coming to the service or going from the service, respectively. The direction MUST be the same as the direction of the message identified by the {message label} property in the {message exchange pattern} of the Interface Operation component being bound by the containing Binding Operation component.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Binding Message Reference component in the {message references} property of a Binding Operation component, the {message label} property MUST be unique. That is, the same message cannot be bound twice within the same operation.

2.12.2 XML Representation of Binding Message Reference Component

```
<definitions>
  <binding>
    <operation>
      <input
            messageLabel="xs:NCName"? >
        <documentation />?
        [ <feature /> | <property /> ]*
      </input>
      <output
            messageLabel="xs:NCName"? >
        <documentation />?
        [ <feature /> | <property /> ]*
      </output>
    </operation>
  </binding>
</definitions>
```

The XML representation for a Binding Message Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `input` or `output` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- One or more *attribute information item*s amongst its [attributes] as follows:
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.12.2.1 messageLabel attribute information item with input or output [owner]** [p.58] .

    If the {message exchange pattern} of the Interface Operation component being bound has only one message with a given value for {direction}, then the `messageLabel` *attribute information item* is optional for the XML representation of the Binding Message Reference component with that {direction}.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]

- Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl". Such *element information item*s are considered to be binding message reference extension elements as described below (see **2.12.2.2 Binding Message Reference extension elements** [p.58]).

2.12.2.1 `messageLabel` *attribute information item* with `input` or `output` [owner]

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel` .
- A [namespace name] which has no value.

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.12.2.2 Binding Message Reference extension elements

Binding Message Reference extension elements are used to provide information specific to a particular message in an operation. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Message Reference component with additional properties and specify the mapping between those properties and the XML representation.

2.12.3 Mapping Binding Message Reference's XML Representation to Component Properties

The mapping between the properties of the Binding Message Reference component (see **2.12.1 The Binding Message Reference Component** [p.56]) and the XML Representation of the `binding` *element information item* (see **2.12.2 XML Representation of Binding Message Reference Component** [p.57]) is as described in Table 2-13 [p.58] .

Table 2-13. Mapping between Binding Message Reference Component Properties and XML Representation

Property	Mapping
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with same {direction} from the {message exchange pattern} of the Interface Operation component being bound, provided there is exactly one such message; otherwise empty.
{direction}	If the [local name] of the *element information item* is `input` then "in", else if the [local name] of the *element information item* is `output` then "out".
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.13 Service

2.13.1 The Service Component

A Service component describes a set of endpoints (see **2.14 Endpoint** [p.62]) at which a particular deployed implementation of the service is provided. The endpoints thus are in effect alternate places at which the service is provided.

Services are named constructs and can be referred to by QName (see **2.18 QName resolution** [p.68]).

The properties of the Service component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {target namespace} REQUIRED. A *wsdls:anyURI* as defined in **2.15.4 anyURI Type** [p.66] .
- {interface} REQUIRED. An Interface component.
- {endpoints} REQUIRED. A non-empty set of Endpoint components.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Service component in the {services} property of a definitions container, the combination of {name} and {target namespace} properties MUST be unique.

2.13.2 XML Representation of Service Component

```
<definitions>
  <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />?
    <endpoint />+
    [ <feature /> | <property /> ]*
  </service>
</definitions>
```

The XML representation for a Service component is an *element information item* with the following Infoset properties:

- A [local name] of `service`
- A [namespace name] of "http://www.w3.org/2004/08/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.13.2.1 name attribute information item with service [owner]** [p.61] .
  - A REQUIRED `interface` *attribute information item* as described below in **2.13.2.2 interface attribute information item with service [owner]** [p.61] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- One or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. One or more *element information item*s from among the following, in any order:
     - One or more `endpoint` *element information item*s (see **2.14.2 XML Representation of Endpoint Component** [p.63]
     - Zero or more `feature` and/or `property` *element information item*s (see **2.7.2 XML Representation of Feature Component** [p.41] and **2.8.2 XML Representation of Property Component** [p.44] , respectively).
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

Note that the XML Schema [*XML Schema: Structures [p.85]*] type of the *element information item* `service` as defined in the WSDL schema MAY be used as the basis for defining new elements which can be used as service references in message exchanges. To enable such reuse, the WSDL schema defines the *attribute information item* `name` as optional in the type of the *element information item* `service` ,

while it is REQUIRED for the *element information item* `service` as indicated above.

Note:

See the primer [*WSDL 2.0 Primer [p.87]*] for more information and examples.

2.13.2.1 `name` *attribute information item* with `service` [owner]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the service.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.13.2.2 `interface` *attribute information item* with `service` [owner]

The `interface` *attribute information item* identifies the interface that the service is an instance of.

The `interface` *attribute information item* has the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] which has no value

The type of the `interface` *attribute information item* is *xs:QName.*.

2.13.3 Mapping Service's XML Representation to Component Properties

The mapping between the properties of the Service component (see **2.13.1 The Service Component** [p.59]) and the XML Representation of the `service` *element information item* (see **2.13.2 XML Representation of Service Component** [p.60]) is as described in Table 2-14 [p.61] .

Table 2-14. Mapping between Service Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*
{interface}	The Interface component resolved to by the actual value of the `interface` *attribute information item*.
{endpoints}	The Endpoint components corresponding to the `endpoint` *element information item*s in [children] if any.
{features}	The set of Feature components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.14 Endpoint

2.14.1 The Endpoint Component

An Endpoint component defines the particulars of a specific endpoint at which a given service is available.

Endpoint components are local to a given Service component; they cannot be referred to by QName.

The properties of the Endpoint component are as follows:

- {name} REQUIRED. A *wsdls:NCName* as defined by **2.15.3 NCName Type** [p.66] .
- {binding} REQUIRED. A named Binding component.
- {address} OPTIONAL. A *wsdls:anyURI* as defined by **2.15.4 anyURI Type** [p.66] . This URI MUST be absolute as defined by [*IETF RFC 2396 [p.84]*]. If present, the value of this attribute represents the network address at which the service indicated by the parent Service component's {interface} property is offered via the binding referred to by the {binding} property.
- {features} OPTIONAL. A set of Feature components.
- {properties} OPTIONAL. A set of Property components.

For each Endpoint component in the {endpoints} property of a Service component, the {binding} property (see **2.14.1 The Endpoint Component** [p.62]) MUST either be a Binding component with an unspecified {interface} property (see **2.9.1 The Binding Component** [p.47] or a Binding component with an {interface} property equal to the {interface} property of the Service component.

For each Endpoint component in the {endpoints} property of a Service component, the {name} property MUST be unique.

2.14.2 XML Representation of Endpoint Component

```
<definitions>
  <service>
    <endpoint
          name="xs:NCName"
          binding="xs:QName"
          address="xs:anyURI"? >
      <documentation />?
      [ <feature /> | <property /> ]*
    </endpoint>
  </service>+
</definitions>
```

The XML representation for a Endpoint component is an *element information item* with the following Infoset properties:

- A [local name] of `endpoint` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.14.2.1 name attribute information item with endpoint [owner]** [p.64] .
  - A REQUIRED `binding` *attribute information item* as described below in **2.14.2.2 binding attribute information item with endpoint [owner]** [p.64] .
  - An OPTIONAL `address` *attribute information item* as described below in **2.14.2.3 address attribute information item with endpoint [owner]** [p.64] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component** [p.41]
     - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component** [p.44]

- Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl". Such *element information item*s are considered to be endpoint extension elements as described below (see **2.14.2.4 Endpoint extension elements** [p.64]).

2.14.2.1 `name` *attribute information item* with `endpoint` [owner]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the endpoint.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name` .
- A [namespace name] which has no value.

The type of the `name` *attribute information item* is *xs:NCName*.

2.14.2.2 `binding` *attribute information item* with `endpoint` [owner]

The `binding` *attribute information item* refers, by QName, to a Binding component

The `binding` *attribute information item* has the following Infoset properties:

- A [local name] of `binding`
- A [namespace name] which has no value

The type of the `binding` *attribute information item* is *xs:QName*.

2.14.2.3 `address` *attribute information item* with `endpoint` [owner]

The `address` *attribute information item* specifies the address of the endpoint.

The `address` *attribute information item* has the following Infoset properties:

- A [local name] of `address`
- A [namespace name] which has no value

The type of the `address` *attribute information item* is *xs:anyURI*.

2.14.2.4 Endpoint extension elements

Endpoint extension elements are used to provide information specific to a particular endpoint in a server. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Endpoint component with additional properties and specify the mapping between those properties and the XML representation.

2.14.3 Mapping Endpoint's XML Representation to Component Properties

The mapping between the properties of the Endpoint component (see **2.14.1 The Endpoint Component** [p.62]) and the XML Representation of the `endpoint` *element information item* (see **2.14.2 XML Representation of Endpoint Component** [p.63]) is as described in Table 2-15 [p.65] .

Table 2-15. Mapping between Endpoint Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item.*
{binding}	The Binding component resolved to by the actual value of the `binding` *attribute information item.*
{address}	The actual value of the `address` *attribute information item* if present, otherwise empty.
{features}	The set of Features components corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item*s in [children], if any.

2.15 Definition of the Simple Types Used in the Component Model

The component model uses a small set of predefined simple types, such as boolean, string, token. In order to avoid introducing a dependency on any particular serialization of the component model, this specification provides its own definition of those types, patterned after [*XML Schema: Datatypes [p.85]*] but independent of it. This allows processors to accept descriptions serialized using a mechanism that is not compatible with [*XML Schema: Datatypes [p.85]*], such as XML 1.1 [*XML 1.1 [p.87]*].

All types defined in this section are formally assigned to the "http://www.w3.org/2004/08/wsdl-simple-types" namespace. All references to them in this specification are made via qualified names that use the *wsdls* prefix. It should be noted though that there is no schema (in the sense of [*XML Schema: Structures [p.85]*]) for that namespace, because the types defined here go beyond the capabilities of XML Schema to describe.

The simple types defined in this specification are:

- *wsdls:string*
- *wsdls:Token*
- *wsdls:NCName*
- *wsdls:anyURI*

- *wsdls:QName*
- *wsdls:boolean*
- *wsdls:int*

All types listed above are such that their value spaces are a superset of the value space of the type with the same name defined by XML Schema [*XML Schema: Datatypes [p.85]*]. In particular, the value space of the *wsdls:string* type is a strict superset of the value space of *xsd:string*, as shown by the one-character string consisting exclusively of the #x0 character.

2.15.1 *string* Type

The value space of the *wsdls:string* type consists of finite-length sequences of characters in the range #x0-#x10FFFF inclusive, where a character is an atomic unit of text as specified by ISO/IEC 10646 [*ISO/IEC 10646 [p.86]*] and Unicode [*Unicode [p.86]*].

2.15.2 *Token* Type

The value space of the *wsdls:Token* type is the subset of the value space of the *wsdls:string* type consisting of strings that do not contain the line feed (#xA), tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces.

2.15.3 *NCName* Type

The value space of the *wsdls:NCName* type is the subset of the value space of the *wsdls:Token* type consisting of tokens that do not contain the space (#x20) and ':' characters.

2.15.4 *anyURI* Type

The value space of the *wsdls:anyURI* type consists of all Uniform Resource Identifiers (URI) as defined by [*IETF RFC 2396 [p.84]*] and amended by [*IETF RFC 2732 [p.85]*].

2.15.5 *QName* Type

The value space of the *wsdls:QName* type consists of the set of 2-tuples whose first component is of type wsdls:anyURI and whose second component is of type wsdls:NCName.

2.15.6 *boolean* Type

The value space of the *wsdls:boolean* type consists of the two distinct values *true* and *false*.

2.15.7 *int* Type

The value space of the *wsdls:int* type consists of the infinite set {...,-2,-1,0,1,2,...} representing the standard mathematical concept of the integer numbers.

2.16 Equivalence of Components

Two component instances of the same type are considered equivalent if, for each property of the first component, there is a corresponding property with an equivalent value on the second component, and the second component has no additional properties.

Instances of properties of the same type are considered equivalent if their values are equivalent.

- For values of a simple type (see **2.15 Definition of the Simple Types Used in the Component Model** [p.65]) this means that they contain the same values. For instance, two string values are equivalent if they contain the same sequence of Unicode characters, as described in [*Character Model for the WWW [p.86]*]
- Values which are references to other components are considered equivalent when they refer to equivalent components (as determined above).
- List-based values are considered equivalent if they have the same length and their elements at corresponding positions are equivalent.
- Finally, set-based values are considered equivalent if they contain corresponding equivalent values, without regard to order.

Extension properties which are not string values, sets of strings or references MUST describe their values' equivalence rules.

Because different top-level components (e.g., Interface, Binding and Service) are required to have different names, it is possible to determine whether two top-level components of a given type are equivalent by examining their {name} and {target namespace} properties.

2.17 Symbol Spaces

This specification defines three symbol spaces, one for each top-level component type (Interface, Binding and Service).

Within a symbol space, all qualified names (that is, the combination of {name} and {target namespace} properties) are unique. Between symbol spaces, the combination of these two properties need not be unique. Thus it is perfectly coherent to have, for example, a binding and an interface that have the same name.

When XML Schema is being used as one of the type systems for a WSDL description, then six other symbol spaces also exist, one for each of: global element declarations, global attribute declarations, named model groups, named attribute groups, type definitions and key constraints, as defined by [*XML Schema: Structures [p.85]*]. Other type systems may define additional symbol spaces.

2.18 QName resolution

In its serialized form WSDL makes significant use of references between components. Such references are made using the Qualified Name, or QName, of the component being referred to. QNames are a tuple, consisting of two parts; a namespace name and a local name. For example, in the case of an Interface component, the namespace name is represented by the {namespace name} property and the local name is represented by the {name} property.

QName references are resolved by looking in the appropriate property of the Definitions component. For example, to resolve a QName of an interface (as referred to by the `interface` *attribute information item* on a binding), the {interfaces} property of the Definitions component would be inspected.

If the appropriate property of the Definitions component does not contain a component with the required QName then the reference is a broken reference. It is an error for a Definitions component to have such broken references.

2.19 Comparing URIs

This specification uses absolute URIs to identify several components (for example, features and properties) and components characteristics (for example, operation message exchange patterns and styles). When such absolute URIs are being compared to determine equivalence (see **2.16 Equivalence of Components** [p.67]) the URIs MUST be compared character-by-character as indicated in [*TAG URI FINDING [p.86]*].

3. Types

```
<definitions>
  <types>
    <documentation />?
    [extension elements]*
  </types>
</definitions>
```

The content of messages and faults may be constrained using type system components. These constraints are based upon a specific data model, and expressed using a particular schema language.

Although a variety of data models can be accommodated (through WSDL extensions), this specification only defines a means of expressing constraints based upon the XML Infoset [*XML Information Set [p.85]*]. Furthermore, although a number of alternate schema languages can be used to constrain the XML Infoset (as long as they support the semantics of either embedding or importing schema), this specification only defines the use of XML Schema [*XML Schema: Structures [p.85]*], [*XML Schema: Datatypes [p.85]*].

Specifically, the {element declarations} and {type definitions} properties of the Definitions component are collections of imported and embedded schema components that describe Infoset *element information item*s.

When extensions are used to enable the use of a non-Infoset data model, or a non-Schema constraint language, the wsdl:required attribute information item MAY be used to require support for that extension.

Note:

Support for the W3C XML Schema Description Language [*XML Schema: Structures [p.85]*],[*XML Schema: Datatypes [p.85]*] is required of all processors.

The schema components contained in the {element declarations} property of **2.1.1 The Definitions Component** [p.11] provide the type system used for Message Reference and Interface Fault components. Message Reference components indicate their structure and content by using the standard *attribute information item*s `element` , or for alternate schema languages in which these concepts do not map well, by using alternative *attribute information item* extensions. Interface Fault components behave similarly. Such extensions should define how they reference type system components. Such type system components MAY appear in additional collection properties on **2.1.1 The Definitions Component** [p.11] .

The schema components contained in the {type definitions} property of **2.1.1 The Definitions Component** [p.11] provide the type system used for constraining the values of properties described by Property components. Extensions in the form of *attribute information item*s can be used to refer to constraints (type definitions or analogous constructs) described using other schema languages or type systems. Such components MAY appear in additional collection properties on **2.1.1 The Definitions Component** [p.11] .

The `types` *element information item* encloses data type definitions, based upon the XML Infoset, used to define messages and has the following Infoset properties:

- A [local name] of `types` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- Zero or more namespace qualified *attribute information item*s in its [attributes] property. The [namespace name] property of such *attribute information item*s MUST NOT be http://www.w3.org/2004/08/wsdl
- Zero or more *element information item*s amongst its [children] as follows:
    - An OPTIONAL `documentation` *element information item* (see **5. Documentation** [p.77]) in its [children] property.
    - Zero or more *element information item*s from among the following, in any order:
        - `xs:import` *element information item*s
        - `xs:schema` *element information item*s
        - Other namespace qualified *element information item*s whose namespace is NOT http://www.w3.org/2004/08/wsdl

3.1 Using W3C XML Schema Description Language

XML Schema MAY be used as the schema language via import or embedding. Each method defines a different *element information item* for use within a `types` *element information item*. All processors MUST support XML Schema type definitions.

A WSDL description MUST NOT refer to XML Schema components in a given namespace unless an `xs:import` and/or `xs:schema` statement for that namespace is present. That is, using the `xs:import` and/or `xs:schema` constructs is a necessary condition for making XML Schema components available to a WSDL description.

3.1.1 Importing XML Schema

Importing an XML Schema uses the syntax and semantics of the `xs:import` mechanism defined by XML Schema [*XML Schema: Structures [p.85]*],[*XML Schema: Datatypes [p.85]*], with some additional restrictions. The schema components defined in the imported schema are available for reference by QName (see **2.18 QName resolution** [p.68]). Note that only components defined in the schema itself and components included by it via `xs:include` are available to WSDL. Specifically, components that the schema imports via `xs:import` are NOT available to WSDL.

A child *element information item* of the `types` *element information item* is defined with the Infoset properties as follows:

- A [local name] of "import".
- A [namespace name] of "http://www.w3.org/2001/XMLSchema".
- One or two *attribute information item*s as follows:
  - A REQUIRED `namespace` *attribute information item* as described below.
  - An OPTIONAL `schemaLocation` *attribute information item* as described below.

3.1.1.1 `namespace` *attribute information item*

The `namespace` *attribute information item* defines the namespace of the element declarations and type definitions imported from the referenced schema. The referenced schema MUST contain a `targetNamespace` *attribute information item* on its `xs:schema` *element information item* and the values of these two *attribute information item*s MUST be identical. It is an error to import a schema that does not have a `targetNamespace` *attribute information item* on its `xs:schema` *element information item*. Such schemas must first be included (using `xs:include`) in a schema that contains a `targetNamespace` *attribute information item* on its `xs:schema` *element information item*, which can then be either imported or inlined in the WSDL document.

The `namespace` *attribute information item* has the following Infoset properties:

- A [local name] of namespace
- A [namespace name] which has no value.

The type of the `namespace` *attribute information item* is *xs:anyURI*.

3.1.1.2 `schemaLocation` *attribute information item*

The `schemaLocation` *attribute information item*, if present, provides a hint to the processor as to where the schema may be located. Caching and cataloging technologies may provide better information than this hint. The `schemaLocation` *attribute information item* has the following infoset properties:

- A [local name] of schemaLocation.
- A [namespace name] which has no value.

The type of the `schemaLocation` *attribute information item* is *xs:anyURI.*

3.1.2 Embedding XML Schema

Embedding an XML schema uses the existing top-level `xs:schema` *element information item* defined by XML Schema [*XML Schema: Structures [p.85]*]. It may be viewed as simply cutting and pasting an existing, stand-alone schema, to a location inside the types *element information item*.

The schema components defined in the embedded schema are available to WSDL for reference by QName (see **2.18 QName resolution** [p.68]). Note that only components defined in the schema itself and components included by it via `xs:include` are available to WSDL. Specifically components that the schema imports via `xs:import` are NOT available to WSDL.

Similarly, components defined in an embedded XML schema are NOT automatically made available to a WSDL description that imported (using `wsdl:import`) the description that embeds the schema (see **4.2 Importing Descriptions** [p.75] for more details). For this reason, it is recommended that XML schema documents intended to be shared across several WSDL descriptions be placed in separate documents and imported using `xs:import` , rather than embedded inside a WSDL document.

Inside an embedded XML schema, the `xs:import` and `xs:include` *element information item*s MAY be used to refer to other XML schemas embedded in the same WSDL description, provided that an appropriate value is specified for their `schemaLocation` *attribute information item*s. The semantics of such *element information item*s are governed solely by the XML Schema specification [*XML Schema: Structures [p.85]*].

The `xs:schema` *element information item* has the following Infoset properties:

- A [local name] of schema.
- A [namespace name] of "http://www.w3.org/2001/XMLSchema".

- A REQUIRED `targetNamespace` *attribute information item*, amongst its [attributes] as described below.
- Additional OPTIONAL *attribute information item*s as specified for the `xs:schema` *element information item* by the XML Schema specification.
- Zero or more child *element information item*s as specified for the `xs:schema` *element information item* by the XML Schema specification.

3.1.2.1 `targetNamespace` *attribute information item*

The `targetNamespace` *attribute information item* defines the namespace of the element declarations and type definitions embedded in its [owner] `xs:schema` *element information item*. WSDL modifies the XML Schema definition of the `xs:schema` *element information item* to make this *attribute information item* required. The `targetNamespace` *attribute information item* has the following infoset properties:

- A [local name] of targetNamespace.
- A [namespace name] which has no value.

The type of the `targetNamespace` *attribute information item* is *xs:anyURI*.

3.1.3 References to Element Declarations and Type Definitions

Whether embedded or imported, the element declarations present in a schema may be referenced from a Message Reference or Interface Fault component. Similarly, regardless of whether they are embedded or imported, the type definitions present in a schema may be referenced from a Property component.

A named, global `xs:element` declaration may be referenced from the `element` *attribute information item* of an `input` , `output` or `fault` *element information item*. The QName is constructed from the `targetNamespace` of the schema and the value of the `name` *attribute information item* of the `xs:element` *element information item*. An `element` *attribute information item* MUST NOT refer to a global `xs:simpleType` or `xs:complexType` definition.

A named, global `xs:simpleType` or `xs:complexType` declaration may be referenced from the `constraint` *attribute information item* of `property` *element information item*. The QName is constructed from the `targetNamespace` of the schema and the value of the `name` *attribute information item* of the `xs:simpleType` or `xs:complexType` *element information item*. A `constraint` *attribute information item* MUST NOT refer to a global `xs:element` definition.

3.2 Using Other Schema Languages

Since it is unreasonable to expect that a single schema language can be used to describe all possible Message Reference, Fault and Property component contents and their constraints, WSDL allows alternate schema languages to be specified via extensibility elements. An extensibility *element information item* MAY appear under the `types` *element information item* to identify the schema language employed, and to locate the schema instance defining the grammar for Message Reference and Interface Fault components or the constraint for Property components. Depending upon the schema language used, an

element information item MAY be defined to allow embedding, if and only if the schema language can be expressed in XML.

A specification of extension syntax for an alternative schema language MUST include the declaration of an *element information item*, intended to appear as a child of the `wsdl:types` *element information item*, which references, names, and locates the schema instance (an "import" *element information item*). The extension specification SHOULD, if necessary, define additional properties of **2.1.1 The Definitions Component** [p.11] (and extensibility attributes) to hold the components of the referenced type system. It is expected that additional extensibility attributes for Message Reference, Interface Fault and Property components will also be defined, along with a mechanism for resolving the values of those attributes to a particular imported type system component.

See **E. Examples of Specifications of Extension Elements for Alternative Schema Language Support.** [p.93] for examples of using other schema languages. These examples reuse the {element declarations} property of **2.1.1 The Definitions Component** [p.11] and the `element` *attribute information item*s of the `wsdl:input` , `wsdl:output` and `wsdl:fault` *element information item*s.

4. Modularizing WSDL descriptions

This specification provides two mechanisms, described in this section, for modularizing WSDL descriptions. These mechanisms help to make WSDL descriptions clearer by allowing separation of the various components of a description. Such separation could be performed according to the level of abstraction of a given set of components, or according to the namespace affiliation required of a given set of components or according to some other grouping such as application applicability.

Both mechanisms work at the level of WSDL components and NOT at the level of XML Information Sets or XML 1.0 serializations.

4.1 Including Descriptions

```
<definitions>
  <include
        location="xs:anyURI" >
    <documentation />?
  </include>
</definitions>
```

The WSDL `include` *element information item* allows for the separation of different components of a service definition, belonging the same target namespace, into independent WSDL documents which can be merged as needed.

The WSDL `include` *element information item* is modeled after the XML Schema `include` *element information item* (see [*XML Schema: Structures [p.85]*], section 4.2.3 "References to schema components in the same namespace"). Specifically, it can be used to include components from WSDL descriptions that share a target namespace with the including description. Components in *directly* included descriptions become part of the component model of the including description. Directly included means that component inclusion is not transitive; components included by one of the included documents are *not* available to the original including document unless the are included directly by that document. The

included components can be referenced by QName. Note that because all WSDL descriptions have a target namespace, no-namespace includes (sometimes known as "chameleon includes") never occur in WSDL.

A mutual include is direct inclusion by one WSDL document of another WSDL document which includes the first. A circular include achieves the same effect with greater indirection (WSDL A includes WSDL B includes WSDL A, for instance). Multiple inclusion of a single WSDL document resolves to a single set of components. Mutual, multiple, and circular includes are explicitly permitted, and do not represent multiple redefinitions of the same components. Multiple inclusion of a single WSDL document has the same meaning as including it only once. Processors are encouraged to keep track of the source of component definitions, so that multiple, mutual, and circular includes do not require establishing identity on a component-by-component basis.

The `include` *element information item* has:

- A [local name] of `include` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `location` *attribute information item* as described below in **4.1.1 location attribute information item with include [owner]** [p.74] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], as follows:
  - An optional `documentation` *element information item* (see **5. Documentation** [p.77]).
  - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

4.1.1 `location` *attribute information item* with `include` [owner]

The `location` *attribute information item* has the following Infoset properties:

- A [local name] of `location` .
- A [namespace name] which has no value.

A `location` *attribute information item* is of type `xs:anyURI` . Its actual value is the location of some information about the namespace identified by the `targetNamespace` *attribute information item* of the containing `definitions` *element information item*.

If the URI indicated by `location` is not dereferenceable or does not resolve to a WSDL document then the processor MUST fail immediately. That is, `include` elements MUST be processed immediately by WSDL processors.

The actual value of the `targetNamespace` *attribute information item* of the included WSDL document MUST match the actual value of the `targetNamespace` *attribute information item* of the `definitions` *element information item* which is the [parent] of the `include` *element information item*.

4.2 Importing Descriptions

```
<definitions>
  <import
        namespace="xs:anyURI"
        location="xs:anyURI"? >
    <documentation />?
  </import>
</definitions>
```

The WSDL `import` *element information item*, like the `include` *element information item* (see **4.1 Including Descriptions** [p.73]) also allows for the separation of the different components of a WSDL description into independent descriptions, but in this case with different target namespaces, which can be imported as needed. This technique helps writing clearer WSDL descriptions by separating the definitions according to their level of abstraction, and maximizes reusability.

The WSDL `import` *element information item* is modeled after the XML Schema `import` *element information item* (see [*XML Schema: Structures [p.85]*], section 4.2.3 "References to schema components across namespaces"). Specifically, it can be used to import components from WSDL descriptions that do not share a target namespace with the importing document. Components in *directly* imported descriptions are part of the component model of the importing description. Directly imported means that component importation is not transitive; components imported by one of the imported documents are *not* available to the original importing document unless the are imported directly by that document. The imported components can be referenced by QName.

Using the `import` construct is a necessary condition for making components from another namespace available to a WSDL description. That is, a WSDL description MUST NOT refer to components in a namespace other than the target namespace unless an import statement for that namespace is present. The same considerations apply to schemas embedded in an imported WSDL description (see **3.1.2 Embedding XML Schema** [p.71]). More explicitly, components defined by an XML schema document embedded inside an imported WSDL description are NOT made available to the importer unless the latter contains an explicit `xs:import` statement to that purpose.

This specification DOES NOT preclude repeating the `import` *element information item* for the same value of the `namespace` *attribute information item* as long as they provide different values for the `location` *attribute information item*. Repeating the `import` *element information item* for the same `namespace` value MAY be used as a way to provide alternate locations to find information about a given namespace.

Furthermore, this specification DOES NOT require the `location` *attribute information item* to be dereferenceable. If it is not dereferenceable then no information about the imported namespace is provided by that `import` *element information item*. It is possible that such lack of information results in QNames in other parts of a WSDL Definitions component to become broken references (see **2.18 QName resolution** [p.68]). Such broken references are not errors of the `imports` *element information item* but rather QName resolution errors which must be detected as described in **2.18 QName resolution** [p.68] .

The `import` *element information item* has the following Infoset properties:

- A [local name] of `import` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `namespace` *attribute information item* as described below in **4.2.1 namespace attribute information item** [p.76] .
  - An OPTIONAL `location` *attribute information item* as described below in **4.2.2 location attribute information item with import [owner]** [p.77] .
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".
- Zero or more *element information item* amongst its [children], as follows:
  - An optional `documentation` *element information item* (see **5. Documentation** [p.77]).
  - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/08/wsdl".

4.2.1 `namespace` *attribute information item*

The `namespace` *attribute information item* has the following Infoset properties:

- A [local name] of `namespace` .
- A [namespace name] which has no value.

The `namespace` *attribute information item* is of type `xs:anyURI` . Its actual value indicates that the containing WSDL document MAY contain qualified references to WSDL definitions in that namespace (via one or more prefixes declared with namespace declarations in the normal way). This value MUST NOT match the actual value of the enclosing WSDL document `targetNamespace` *attribute information item*. If the import statement results in the import of a WSDL document then the actual value of the `namespace` *attribute information item* MUST be identical to the actual value of the imported WSDL document's `targetNamespace` *attribute information item*.

4.2.2 `location` *attribute information item* with `import` [owner]

The `location` *attribute information item* has the following Infoset properties:

- A [local name] of `location` .
- A [namespace name] which has no value.

The `location` *attribute information item* is of type `xs:anyURI` . Its actual value is the location of some information about the namespace identified by the `namespace` *attribute information item*.

The `location` *attribute information item* is optional. This allows WSDL components to be constructed from information other than serialized XML 1.0. It also allows the development of WSDL processors that have *a priori* (i.e., built-in) knowledge of certain namespaces.

5. Documentation

```
<documentation>
  [extension elements]*
</documentation>
```

WSDL uses the optional `documentation` *element information item* as a container for human readable and/or machine processable documentation. The content of the *element information item* is arbitrary *character information items* and *element information item*s ("mixed" content in XML Schema[*XML Schema: Structures [p.85]*]). The `documentation` *element information item* is allowed inside any WSDL *element information item*.

The `documentation` *element information item* has:

- A [local name] of `documentation` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- Zero or more *attribute information item*s in its [attributes] property.
- Zero or more child *element information item*s in its [children] property.
- Zero or more *character information item*s in its [children] property.

6. Language Extensibility

In addition to extensibility implied by the Feature and Property components described above, the schema for WSDL has a two-part extensibility model based on namespace-qualified elements and attributes. An extension is identified by the qname consisting of its namespace URI and its element name. The meaning of an extension SHOULD be defined (directly or indirectly) in a document that is available at its namespace URI.

6.1 Element based Extensibility

WSDL allows extensions to be defined in terms of *element information item*s. Where indicated herein, WSDL allows namespace-qualified *element information item*s whose [namespace name] is NOT "http://www.w3.org/2004/08/wsdl" to appear among the [children] of specific *element information item*s whose [namespace name] is "http://www.w3.org/2004/08/wsdl". Such *element information item*s MAY be used to annotate WSDL constructs such as interface, operation, etc.

It is expected that extensions will want to add to the existing properties of components in the component model. The specification for an extension *element information item* should include definitions of any such properties and the mapping between the XML representation of the extension and the properties in the component model.

The WSDL schema also defines a base type for use by extensibility elements. Example 6-1 [p.78] shows the type definition. The use of this type as a base type is optional. The element declarations which serve as the heads of the defined substitution groups are all of type "xs:anyType".

Extensibility elements are commonly used to specify some technology-specific binding. They allow innovation in the area of network and message protocols without having to revise the base WSDL specification. WSDL recommends that specifications defining such protocols also define any necessary WSDL extensions used to describe those protocols or formats.

Example 6-1. Base type for extensibility elements

```
<xs:complexType name='ExtensibilityElement' abstract='true' >
  <xs:attribute ref='wsdl:required' use='optional' />
</xs:complexType>
```

6.1.1 Mandatory extensions

Extension elements can be marked as mandatory by annotating them with a `wsdl:required` *attribute information item* (see **6.1.2 required attribute information item** [p.79]) with a value of "true". A mandatory extension is an extension that MAY change the meaning of the element to which it is attached, such that the meaning of that element is no longer governed by this specification. Instead, the meaning of an element containing a mandatory extension is governed by the meaning of that extension. Thus, the definition of the element's meaning is *delegated* to the specification that defines the extension.

An extension that is NOT marked as mandatory MUST NOT invalidate the meaning of any part of the WSDL document. Thus, a NON-mandatory extension merely provides additional description of capabilities of the service. This specification does not provide a mechanism to mark extension attributes as being required. Therefore, all extension attributes are NON-mandatory.

Note:

A mandatory extension is considered mandatory because it has the ability to change the meaning of the element to which it is attached. Thus, the meaning of the element may not be fully understood without understanding the attached extension. A NON-mandatory extension, on the other hand, can be safely ignored without danger of misunderstanding the rest of the WSDL document.

If a WSDL document declares an extension, Feature or Property as optional (i.e., NON-mandatory), then the provider agent MUST NOT assume that the requester agent supports that extension, Feature or Property, *unless* the provider agent knows (through some other means) that the requester agent has in fact elected to engage and support that extension, Feature or Property.

On the other hand, a requester agent MAY engage an extension, Feature or Property that is declared as optional in the WSDL document. Therefore, the provider agent MUST support every extension, Feature or Property that is declared as optional in the WSDL document, in addition to supporting every extension, Feature or Property that is declared as mandatory.

Note:

If finer-grain, direction-sensitive control of extensions, Features or Properties is desired, then such extensions, Features or Properties may be designed in a direction-sensitive manner (from requester or from provider) so that either direction may be separately marked required or optional. For example, instead of defining a single extension that governs both directions, two extensions could be defined -- one for each direction.

6.1.2 `required` *attribute information item*

WSDL provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `required` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl".
- A [specified] property with a value of "true".

The type of the `required` *attribute information item* is *xs:boolean*.

6.2 Attribute-based Extensibility

WSDL allows qualified *attribute information item*s whose [namespace name] is NOT "http://www.w3.org/2004/08/wsdl" to appear on any *element information item* whose namespace name IS "http://www.w3.org/2004/08/wsdl". Such *attribute information item*s can be used to annotate WSDL constructs such as interfaces, bindings, etc.

WSDL does not provide a mechanism for marking extension *attribute information item*s as mandatory.

6.3 Extensibility Semantics

As indicated above, it is expected that the presence of extensibility elements and attributes will result in additional properties appearing in the component model.

The presence of an optional extensibility element or attribute MAY therefore augment the semantics of a WSDL document in ways that do not invalidate the existing semantics. However, the presence of a mandatory extensibility element MAY alter the semantics of a WSDL document in ways that invalidate the existing semantics.

Note:

Authors of extensibility elements should avoid altering the existing semantics in ways that are likely to confuse users.

7. Locating WSDL Documents

As an XML vocabulary, WSDL documents or fragments or references to WSDL components (via QNames) MAY appear within other XML documents. In such scenarios it could be necessary to provide some hints on where additional WSDL information for a given namespace can be found in order to help with QName resolution **2.18 QName resolution** [p.68] .

This specification defines a global attribute, `wsdlLocation` in the namespace "http://www.w3.org/2004/08/wsdl-instance" for this purpose (hereafter referred to as "wsdli:wsdlLocation"). This global attribute MAY appear on any XML element which allows attributes from other namespaces to occur. It MUST NOT appear on a `wsdl:definitions` element or any of its children/descendants.

7.1 `wsdli:wsdlLocation` *attribute information item*

WSDL provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `wsdlLocation` .
- A [namespace name] of "http://www.w3.org/2004/08/wsdl-instance".

The type of the `wsdlLocation` *attribute information item* is a list *xs:anyURI*. Its actual value MUST be a list of pairs of URIs; where the first URI of a pair, which MUST be an absolute URI as defined in [*IETF RFC 2396 [p.84]*], indicates a WSDL namespace name, and, the second a hint as to the location of a WSDL document defining WSDL components for that namespace name. The second URI of a pair MAY be absolute or relative.

8. Conformance

8.1 Document Conformance

An *element information item* whose namespace name is "http://www.w3.org/2004/08/wsdl" and whose local part is `definitions` conforms to this specification if it conforms to the XML Schema for that element as defined by this specification family and additionally adheres to all the constraints contained in this specification.

Note that the WSDL language is defined in terms of the component model defined by this specification. As such, it is explicitly NOT a conformance requirement to be able to process documents encoded in a particular version of XML, in particular XML 1.1 [*XML 1.1 [p.87]*].

8.2 XML Information Set Conformance

This specification conforms to the [*XML Information Set [p.85]*]. The following information items MUST be present in the input infosets to enable correct processing of WSDL documents:

- *Document Information Items* with *children* and *base URI* properties.
- *Element Information Items* with *namespace name*, *local name*, *children*, *attributes*, *base URI* and *parent* properties.
- *Attribute Information Items* with *namespace name*, *local name* and *normalized value* properties.
- *Character Information Items* with *character code*, *element content whitespace* and *parent* properties.

8.3 Processor Conformance

This section defines a class of conformant WSDL processors that are intended to act on behalf of a party that wishes to make use of a Web service (i.e., the requester entity or requester agent), rather than the party that implements the Web service (i.e., the provider entity or provider agent).

An extension element is said to be *processed* if the WSDL processor decides (through whatever means) that its parent (an *element information item* in the "http://www.w3.org/2004/08/wsdl" namespace) will be processed. Note that it is possible for WSDL processors to process only a subset of a given WSDL document. For instance, a tool may wish to focus on interfaces and operations only, and ignore bindings.

A conformant WSDL processor MUST adhere to the following rules:

- Except as noted below for mandatory extensions, a conformant WSDL processor MUST accept any legal WSDL document as defined by this specification.
- A conformant WSDL processor MUST fault if a portion of a WSDL document is illegal according to this specification and the WSDL processor attempts to process that portion.
- A conformant WSDL processor MUST support at least XML Schema as a type system language.
- A conformant WSDL processor MUST fail if it processes an element containing a wsdl:include statement having a URI that is not dereferenceable to a legal WSDL document.
- If a mandatory extension (i.e., a mandatory element, feature or property) is processed, a conformant WSDL processor MUST either agree to fully abide by all the rules and semantics signaled by that extension, or immediately cease processing (fault). In particular, if the WSDL processor does not recognize the extension, it MUST fault. If the WSDL processor recognizes the extension, and determines that the extension in question is incompatible with any other aspect of the document (including other required extensions), it MUST fault.
- A conformant WSDL processor MAY safely ignore a NON-mandatory extension or feature that it does not recognize or that it does not choose to implement.

Note:

If a WSDL document declares an extension or feature as optional, then if that extension or feature could apply to messages sent by the provider agent as well, then the provider agent MUST NOT send any messages that requires the requester agent to support that extension or feature. The requestor, on the othe hand, MAY engage that extension or feature in messages it sends to the provider.

If finer-grain control of extensions and features is desired then such extensions and features must be designed in a direction (from requestor or from provider) sensitive manner so that any direction may be marked required or optional.

- If a wsdl:definitions element is processed, a conformant WSDL processor MUST also process the wsdl:import, wsdl:include, and wsdl:types children of that element.
- If a wsdl:interface element is processed, a conformant WSDL processor MUST also process the wsdl:operation, wsdl:fault, wsdl:feature, and wsdl:property children of that element.
- If a wsdl:binding element is processed, a conformant WSDL processor MUST also process the wsdl:operation, wsdl:fault, wsdl:feature, and wsdl:property children of that element.
- If a wsdl:operation element is processed, a conformant WSDL processor MUST also process the wsdl:input, wsdl:output, wsdl:infault, wsdl:outfault, wsdl:feature, and wsdl:property children of that element.
- If a wsdl:property element is processed, a conformant WSDL processor MUST also process the wsdl:value and wsdl:constraint children of that element.

9. XML Syntax Summary (Non-Normative)

```
<definitions targetNamespace="xs:anyURI" >
  <documentation />?

  <import namespace="xs:anyURI" location="xs:anyURI"? >
    <documentation />?
  </import>*

  <include location="xs:anyURI" >
    <documentation />?
  </include>*

  <types>
    <documentation />?
  </types>

  <interface name="xs:NCName" extends="list of xs:QName"? styleDefault="list of xs:anyURI"? >
    <documentation />?

    <fault name="xs:NCName" element="xs:QName"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </fault>*
```

```
  <operation name="xs:NCName" pattern="xs:anyURI" style="list of xs:anyURI"? safe="xs:boolean"? >
    <documentation />?

    <input messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </input>*

    <output messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </output>*

    <infault ref="xs:QName" messageLabel="xs:NCName"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </infault>*

    <outfault ref="xs:QName" messageLabel="xs:NCName"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </outfault>*

    <feature ... />*

    <property ... />*
  </operation>*

  <feature uri="xs:anyURI" required="xs:boolean"? >
    <documentation />?
  </feature>*

  <property uri="xs:anyURI" required="xs:boolean"? >
    <documentation />?

    <value> xs:anyType </value>?

    <constraint> xs:QName </constraint>?
  </property>*
</interface>*

<binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI" >
  <documentation />?

  <fault ref="xs:QName" >
    <documentation />?

    <feature ... />*

    <property ... />*
  </fault>*

  <operation ref="xs:QName" >
```

```
      <documentation />?

      <input messageLabel="xs:NCName"? >
        <documentation />?

        <feature ... />*

        <property ... />*
      </input>*

      <output messageLabel="xs:NCName"? >
        <documentation />?

        <feature ... />*

        <property ... />*
      </output>*

      <feature ... />*

      <property ... />*
    </operation>*

    <feature ... />*

    <property ... />*
  </binding>*

  <service name="xs:NCName" interface="xs:QName" >
    <documentation />?

    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? >
      <documentation />?

      <feature ... />*

      <property ... />*
    </endpoint>*

    <feature ... />*

    <property ... />*
  </service>*
</definitions>
```

10. References

10.1 Normative References

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

[IETF RFC 2396]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt.

[IETF RFC 2732]
Format for Literal IPv6 Addresses in URL's., R. Hinden, B. Carpenter, L. Masinter Authors. Internet Engineering Task Force, December 1999. Available at http://www.ietf.org/rfc/rfc2732.txt.

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Editors. World Wide Web Consortium, 10 February 1998, revised 6 October 2000. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version of XML 1.0 is available at http://www.w3.org/TR/REC-xml.

[XML Information Set]
XML Information Set, J. Cowan and R. Tobin, Editors. World Wide Web Consortium, 24 October 2001. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2001/REC-xml-infoset-20011024. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset.

[XML Namespaces]
Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Editors. World Wide Web Consortium, 14 January 1999. This version of the XML Information Set Recommendation is http://www.w3.org/TR/1999/REC-xml-names-19990114. The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names.

[XML Schema: Structures]
XML Schema Part 1: Structures, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 1 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-1-20010502. The latest version of XML Schema Part 1 is available at http://www.w3.org/TR/xmlschema-1.

[XML Schema: Datatypes]
XML Schema Part 2: Datatypes, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2.

[RFC 3023]
IETF "RFC 3023: XML Media Types", M. Murata, S. St. Laurent, D. Kohn, July 1998. (See *http://www.ietf.org/rfc/rfc3023.txt*.)

[WSDL MediaType]
IETF Internet Draft "The 'application/wsdl+xml' media type", @@@. (Work to be done once we have consensus on the media type).

[WSDL 2.0 Bindings]
Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings, H. Haas, P. Le HÃ©garet, J-J. Moreau, D. Orchard, J. Schlimmer, Editors. World Wide Web Consortium, 3 August 2004. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings" Specification is available at http://www.w3.org/TR/2004/WD-wsdl20-bindings-20040803. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings" is available at http://www.w3.org/TR/wsdl20-bindings.

[WSDL 2.0 Predefined Extensions]
Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions, M. Gudgin, A. Lewis, and J. Schlimmer, Editors. World Wide Web Consortium, 3 August 2004. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions" Specification is available at http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803. The

latest version of "Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions" is available at http://www.w3.org/TR/wsdl20-extensions.

[WSDL 2.0 RDF Mapping]
Web Services Description (WSDL) Version 2.0: RDF Mapping, XYZ, Editors. World Wide Web Consortium, 3 August 2004. This version of the "Web Services Description Version 2.0: RDF Mapping" Specification is available at http://www.w3.org/TR/2004/WD-wsdl20-extensions-20040803. The latest version of "Web Services Description Version 2.0: RDF Mapping" is available at http://www.w3.org/TR/wsdl20-extensions.

[Character Model for the WWW]
Character Model for the World Wide Web 1.0: Fundamentals, M. Durst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, Editors. W3C Working Draft, 25 February 2004. Available at http://www.w3.org/TR/charmod/.

[TAG URI FINDING]
TAG Finding on URI Comparison, X. Foo, Y. Bar, Authors. W3C Technical Architecture Group, Month, Year. Draft available at http://www.textuality.com/tag/uri-comp-4.

[Web Architecture]
Architecture of the World Wide Web, First Edition, Ian Jacobs, Editor. W3C Technical Architecture Group, December, 2003. Available at http://www.w3.org/TR/2003/WD-webarch-20031209/.

[ISO/IEC 10646]
ISO/IEC 10646-1:2000. Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane (See http://www.iso.ch for the latest version.)

[Unicode]
The Unicode Consortium, The Unicode Standard, Version 4 , ISBN 0-321-18578-1, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information on versions of the standard and of the Unicode Character Database).

10.2 Informative References

[IETF RFC 2045]
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N. Freed, N. Borenstein, Authors. Internet Engineering Task Force, November 1996. Available at http://www.ietf.org/rfc/rfc2045.txt.

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

[SOAP 1.1]
Simple Object Access Protocol (SOAP) 1.1, D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer, Editors. World Wide Web Consortium, 8 May 2000. This version of the Simple Object Access Protocol 1.1 Note is http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[SOAP 1.2 Part 1: Messaging Framework]
SOAP Version 1.2 Part 1: Messaging Framework, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, Editors. World Wide Web Consortium, 24 June 2003. This version of

the "SOAP Version 1.2 Part 1: Messaging Framework" Recommendation is http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. The latest version of "SOAP Version 1.2 Part 1: Messaging Framework" is available at http://www.w3.org/TR/soap12-part1/.

[XML Linking]
XML Linking Language (XLink) Version 1.0, S. DeRose, E. Maler, D. Orchard, Editors. World Wide Web Consortium, 27 June 2001. This version of the XML Linking Language 1.0 Recommendation is http://www.w3.org/TR/2001/REC-xlink-20010627. The latest version of XML Linking Language 1.0 is available at http://www.w3.org/TR/xlink.

[WSDL 1.1]
Web Services Description Language (WSDL) 1.1, E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Authors. World Wide Web Consortium, 15 March 2002. This version of the Web Services Description Language 1.1 Note is http://www.w3.org/TR/2001/NOTE-wsdl-20010315. The latest version of Web Services Description Language 1.1 is available at http://www.w3.org/TR/wsdl.

[WSDL 2.0 Primer]
Web Services Description (WSDL) Version 2.0: Primer, K. Sankar, K. Liu, D. Booth, Editors. World Wide Web Consortium, 3 August 2004. The editors' version of the Web Services Description Version 2.0: Primer document is available from http://www.w3.org/2002/ws/desc/.

[WSD Requirements]
Web Services Description Requirements, J. Schlimmer, Editor. World Wide Web Consortium, 28 October 2002. This version of the Web Services Description Requirements document is http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028. The latest version of Web Services Description Requirements is available at http://www.w3.org/TR/ws-desc-reqs.

[XPointer Framework]
XPointer Framework,Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh, Editors. World Wide Web Consortium, 22 November 2002. This version of the XPointer Framework Proposed Recommendation is http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ The latest version of XPointer Framework is available at http://www.w3.org/TR/xptr-framework/.

[XML 1.1]
Extensible Markup Language (XML) 1.1 , T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, Francois Yergau, and John Cowan, Editors. World Wide Web Consortium, 04 February 2004, edited in place 15 April 2004. This version of the XML 1.1 Recommendation is http://www.w3.org/TR/2004/REC-xml-20040204. The latest version of XML 1.1 is available at http://www.w3.org/TR/xml11.

A. The "application/wsdl+xml" Media Type

Editorial note: JJM	20021107
This was lifted from the SOAP 1.2 specification, and needs to be edited to reflect WSDL's own requirements. For example, the WG has not reached consensus on whether to use "text/xml", "text/wsdl+xml" or "application/wsdl+xml".	

This appendix defines the "application/wsdl+xml" media type which can be used to describe WSDL 2.0 documents serialized as XML. It is referenced by the corresponding IANA registration document [*WSDL MediaType [p.85]*].

A.1 Registration

MIME media type name:

application

MIME subtype name:

wsdl+xml

Required parameters:

none

Optional parameters:

charset

This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [*RFC 3023 [p.85]*].

Encoding considerations:

Identical to those of "application/xml" as described in [*RFC 3023 [p.85]*], section 3.2, as applied to the WSDL document infoset.

Security considerations:

See section **A.2 Security considerations** [p.89] .

Interoperability considerations:

There are no known interoperability issues.

Published specification:

This document and [*WSDL 2.0 Bindings [p.85]*].

Applications which use this media type:

No known applications currently use this media type.

Additional information:

File extension:

WSDL documents are not required or expected to be stored as files.

Fragment identifiers:

Either a syntax identical to that of "application/xml" as described in [*RFC 3023 [p.85]*], section 5 or the syntax defined in **C. URI References for WSDL constructs** [p.90] .

Base URI:

As specified in [*RFC 3023 [p.85]*], section 6.

Macintosh File Type code:

TEXT

Person and email address to contact for further information:

@@@ <@@@@>

Intended usage:

COMMON

Author/Change controller:

The WSDL 2.0 specification set is a work product of the World Wide Web Consortium’s Web Service Description Working Group. The W3C has change control over these specifications.

A.2 Security considerations

Editorial note: JJM	20021107
Are there any security considerations other than the standard ones.	

This media type uses the "+xml" convention, it shares the same security considerations as described in [*RFC 3023 [p.85]*], section 10.

B. Acknowledgements (Non-Normative)

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

Members of the Working Group are (at the time of writing, and by alphabetical order): David Booth (W3C), Allen Brookes (Rogue Wave Softwave), Helen Chen (Agfa-Gevaert N. V.), Roberto Chinnici (Sun Microsystems), Ugo Corda (SeeBeyond), Glen Daniels (Sonic Software), Paul Downey (British Telecommunications), Youenn Fablet (Canon), Martin Gudgin (Microsoft Corporation), Hugo Haas (W3C), Hao He (The Thomson Corporation), Tom Jordahl (Macromedia), Jacek Kopecky (Digital Enterprise Research Institute (DERI)), Amelia Lewis (TIBCO Software, Inc.), Kevin Canyang Liu (SAP), Jonathan Marsh (Microsoft Corporation), Peter Madziak (Agfa-Gevaert N. V.), Josephine Micallef (SAIC - Telcordia Technologies), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Cyclone Commerce), Jean-Jacques Moreau (Canon), Mark Nottingham (BEA Systems, Inc.), David Orchard (BEA Systems, Inc.), Bijan Parsia (University of Maryland), Arthur Ryman (IBM), Adi Sakala (IONA Technologies),

Jeffrey Schlimmer (Microsoft Corporation), Igor Sedukhin (Computer Associates), Jerry Thrasher (Lexmark), William Vambenepe (Hewlett-Packard Company), Asir Vedamuthu (webMethods, Inc.), Sanjiva Weerawarana (IBM), Ãmit YalÃ§Ä±nalp (Oracle Corporation), Prasad Yendluri (webMethods, Inc.).

Previous members were: Lily Liu (webMethods, Inc.), Don Wright (Lexmark), Joyce Yang (Oracle Corporation), Daniel Schutzer (Citigroup), Dave Solo (Citigroup), Stefano Pogliani (Sun Microsystems), William Stumbo (Xerox), Stephen White (SeeBeyond), Barbara Zengler (DaimlerChrysler Research and Technology), Tim Finin (University of Maryland), Laurent De Teneuille (L'Echangeur), Johan Pauhlsson (L'Echangeur), Mark Jones (AT&T), Steve Lind (AT&T), Sandra Swearingen (U.S. Department of Defense, U.S. Air Force), Philippe Le HÃ©garet (W3C), Jim Hendler (University of Maryland), Dietmar Gaertner (Software AG), Michael Champion (Software AG), Don Mullen (TIBCO Software, Inc.), Steve Graham (Global Grid Forum), Steve Tuecke (Global Grid Forum), Michael Mahan (Nokia), Bryan Thompson (Hicks & Associates), Ingo Melzer (DaimlerChrysler Research and Technology), Sandeep Kumar (Cisco Systems), Alan Davies (SeeBeyond), Jacek Kopecky (Systinet), Mike Ballantyne (Electronic Data Systems), Mike Davoren (W. W. Grainger), Dan Kulp (IONA Technologies), Mike McHugh (W. W. Grainger), Michael Mealling (Verisign), Waqar Sadiq (Electronic Data Systems), Yaron Goland (BEA Systems, Inc.).

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

C. URI References for WSDL constructs (Non-Normative)

This appendix provides a syntax for URI references for named components found in a WSDL document. This includes the top level components: interface, binding and service and the subordinate components: operation, fault, and endpoint. The URI references are easy to understand and compare, while imposing no burden on the WSDL author.

C.1 WSDL URIs

There are two main cases for WSDL URIs:

- the URI of a WSDL document
- the URI of a WSDL namespace

The URI of a WSDL document can be dereferenced to give a resource representation that contributes component definitions to a single WSDL namespace. If the media type is set to the WSDL media type, then the fragment identifiers can be used to identify the main components that are defined in the document.

However, in keeping with the recommendation in **2.1.1 The Definitions Component** [p.11] that the namespace URI be dereferencible to a WSDL document, this appendix specifies the use of the namespace URI with the WSDL fragment identifiers to form a URI-reference.

C.2 Fragment Identifiers

The following fragment identifier syntax is compliant with the [*XPointer Framework [p.87]*].

The URI in a URI-reference for a WSDL component is the {target namespace} property of either the component itself, in the case of interfaces, bindings, and services, or the {target namespace} property of an ancestor component. The URI provided by the {target namespace} property is combined with a fragment identifier, where the fragment identifier is constructed from the {name} property of the component and the {name} properties of its ancestors as a path according to Table C-1 [p.91] . In that table the first column gives the name of the WSDL component as the [local name] of the *element information item* that represents that construct in a WSDL document. Columns two and three populate the variables x and y respectively. These variables are then used to construct the fragment in column four.

Table C-1. Rules for determining fragments for WSDL constructs

Construct	x	y	Fragment
interface	{name} property of interface	n/a	`interface(x)`
operation	{name} property of operation	{name} property of parent interface	`operation(y/x)`
fault	{name} property of fault	{name} property of parent interface	`fault(y/x)`
binding	{name} property of binding	n/a	`binding(x)`
service	{name} property of service	n/a	`service(x)`
endpoint	{name} property of endpoint	{name} property of parent service	`endpoint(y/x)`

Note that the above rules are defined in terms of component properties rather the XML Infoset representation of the component model.

C.3 Extension Elements

WSDL has an open content model. It is therefore possible for an extension to define new components. The XPointer Framework scheme for components added by extensions is:

```
extension(extension-namespace, extension-specific-syntax)
```

where extension-namespace is the namespace that identifies the extension, e.g. for SOAP the namespace is http://www.w3.org/2003/06/wsdl/soap12, and extension-specific-syntax is defined by the extension. The owner of the extension must define any components contributed by the extension and a syntax for identifying them.

C.4 Example

Consider the following WSDL located at http://example.org/TicketAgent.wsdl:

Example C-1. URI References - Example WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/2004/08/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2004/08/wsdl wsdl20.xsd">

    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
                   namespace="http://example.org/TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights" pattern="http://www.w3.org/2004/08/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
        </wsdl:operation>

        <wsdl:operation name="reserveFlight" pattern="http://www.w3.org/2004/08/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:reserveFlightRequest"/>
            <wsdl:output element="xsTicketAgent:reserveFlightResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:definitions>
```

Its conceptual elements have the following URI-references:

Example C-2. URI References - Example URIs

```
http://example.org/TicketAgent.wsdl20#interface(TicketAgent)
http://example.org/TicketAgent.wsdl20#operation(TicketAgent/listFlights)
http://example.org/TicketAgent.wsdl20#operation(TicketAgent/reserveFlight)
```

D. Migrating from WSDL 1.1 to WSDL 2.0 (Non-Normative)

This section will attempt to document some of the migration concerns of going from WSDL 1.1 to WSDL 2.0. We do not claim that all migration problems will be addressed here.

D.1 Operation Overloading

WSDL 1.1 supported operation overloading and WSDL 2.0 removes it. This section will provide some rationale for it and provide hints on how to work around some scenarios.

D.2 PortTypes

Port types have been renamed to interfaces. We now have interface inheritance.

D.3 Ports

Ports have been renamed to endpoints.

E. Examples of Specifications of Extension Elements for Alternative Schema Language Support. (Non-Normative)

E.1 DTD

A DTD may be used as the schema language for WSDL. It may not be embedded; it must be imported. A namespace must be assigned. DTD types appear in the {element declarations} property of **2.1.1 The Definitions Component** [p.11] and may be referenced from the `wsdl:input` , `wsdl:output` and `wsdl:fault` elements using the `element` *attribute information item*.

The prefix, dtd, used throughout the following is mapped to the namespace URI "http://www.example.org/dtd/".

The `dtd:import` *element information item* references an external Document Type Definition, and has the following infoset properties:

- A [local name] of import.
- A [namespace name] of "http://www.example.org/dtd".
- One or two *attribute information item*s, as follows:
  - A REQUIRED `namespace` *attribute information item* as described below.
  - An OPTIONAL `location` *attribute information item* as described below.

E.1.1 `namespace` *attribute information item*

The `namespace` *attribute information item* sets the namespace to be used with all imported element definitions described in the DTD. It has the following infoset properties:

- A [local name] of namespace.
- A [namespace name] which has no value.

The type of the `namespace` *attribute information item* is *xs:anyURI*.

The WSDL author should ensure that a prefix is associated with the namespace at the proper scope (probably document scope).

E.1.2 `location` *attribute information item*

The `location` *attribute information item*, if present, provides a hint to the processor as to where the DTD may be located. Caching and cataloging technologies may provide better information than this hint. The `location` *attribute information item* has the following infoset properties:

- A [local name] of location.
- A [namespace name] which has no value.

The type of the `location` *attribute information item* is *xs:anyURI*.

E.1.3 References to Element Definitions

The `element` *attribute information item* MUST be used when referring to an element definition (<!ELEMENT>) from a Message Reference component; referring to an element definition from a Interface Fault component is similar. The value of the element definition MUST correspond to the content of the `namespace` *attribute information item* of the `dtd:import` *element information item*. The local name part must correspond to an element defined in the DTD.

Note that this pattern does not attempt to make DTDs namespace-aware. It applies namespaces externally, in the import phase.

E.2 RELAX NG

A RELAX NG schema may be used as the schema language for WSDL. It may be embedded or imported; import is preferred. A namespace must be specified; if an imported schema specifies one, then the [actual value] of the `namespace` *attribute information item* in the `import` *element information item* must match the specified namespace. RELAX NG provides both type definitions and element declarations, the latter appears in the {element declarations} property of **2.1.1 The Definitions Component** [p.11] respectively. The following discussion supplies the prefix rng which is mapped to the URI "http://www.relaxng.org/ns/structure/1.0".

E.2.1 Importing RELAX NG

Importing a RELAX NG schema uses the rng:include mechanism defined by RNG, with restrictions on its syntax and semantics. A child *element information item* of the `types` *element information item* is defined with the Infoset properties as follows:

- A [local name] of include.
- A [namespace name] of "http://www.relaxng.org/ns/structure/1.0".

- Two *attribute information item*s as follows:
  - A REQUIRED `ns` *attribute information item* as described below.
  - An OPTIONAL `href` *attribute information item* as described below.
  - Additional *attribute information item*s as defined by the RNG specification.

Note that WSDL restricts the `rng:include` *element information item* to be empty. That is, it cannot redefine `rng:start` and `rng:define` *element information item*s; it may be used solely to import a schema.

E.2.1.1 `ns` *attribute information item*

The `ns` *attribute information item* defines the namespace of the type and element definitions imported from the referenced schema. If the referenced schema contains an `ns` *attribute information item* on its `grammar` *element information item*, then the values of these two *attribute information item*s must be identical. If the imported grammar does not have an `ns` *attribute information item* then the namespace specified here is applied to all components of the schema as if it did contain such an *attribute information item*. The `ns` *attribute information item* contains the following Infoset properties:

- A [local name] of ns.
- A [namespace name] which has no value.

The type of the `ns` *attribute information item* is *xs:anyURI*.

E.2.1.2 `href` *attribute information item*

The `href` *attribute information item* must be present, according to the rules of the RNG specification. However, WSDL allows it to be empty, and considers it only a hint. Caching and cataloging technologies may provide better information that this hint. The `href` *attribute information item* has the following Infoset properties:

- A [local name] of href.
- A [namespace name] which has no value.

The type of the `href` *attribute information item* is *xs:anyURI*.

E.2.2 Embedding RELAX NG

Embedding an RNG schema uses the existing top-level `rng:grammar` *element information item*. It may be viewed as simply cutting and pasting an existing, stand-alone schema to a location inside the `wsdl:types` *element information item*. The `rng:grammar` *element information item* has the following Infoset properties:

- A [local name] of grammar.
- A [namespace name] of "http://www.relaxng.org/ns/structure/1.0".
- A REQUIRED `ns` *attribute information item*s as described below.
- Additional *attribute information item*s as specified for the `rng:grammar` *element information item* in the RNG specification.
- Child *element information item*s as specified for the `rng:grammar` *element information item* in the RNG specification.

E.2.2.1 `ns` *attribute information item*

The `ns` *attribute information item* defines the namespace of the type and element definitions embedded in this schema. WSDL modifies the RNG definition of the `rng:grammar` *element information item* to make this *attribute information item* required. The `ns` *attribute information item* has the following infoset properties:

- A [local name] of ns.
- A [namespace name] which has no value.

The type of the `ns` *attribute information item* is *xs:anyURI*.

E.2.3 References to Element Declarations

Whether embedded or imported, the element definitions present in a schema may be referenced from a Message Reference or Interface Fault component.

A named rng:define definition MUST NOT be referenced from the Message Reference or Interface Fault components.

A named Relax NG element declaration MAY be referenced from a Message Reference or Interface Fault component. The QName is constructed from the namespace (`ns` *attribute information item*) of the schema and the content of the `name` *attribute information item* of the `element` *element information item* An `element` *attribute information item* MUST NOT be used to refer to an `rng:define` *element information item*.

F. Part 1 Change Log (Non-Normative)

F.1 WSDL Specification Changes

Date	Author	Description
20040802	RRC	Removed paragraph added per resolution of issue 211 (undone per action item 5 of the 2004-07-29 concall).

20040802	RRC	Added clarification on the meaning of required language extensions.
20040802	RRC	Added operation name requirement to the Interface component section.
20040802	RRC	Added introductory text for the Property Component (per action item 2 of the 2004-07-29 concall).
20040727	RRC	Made the Property component independent of XML Schema (issue 248).
20040727	SW	Issue 243 text
20040727	SW	Incorporated Paul's words for issue 235
20040727	SW	Added MarkN's text for issue 211
20040727	SW	Added note to processor conf rules for optional extensions and features about what optional means.
20040727	SW	Removed contentious area ed note thing per decision to do those via minority opinions.
20040722	HH	Defined wsdls:int for http:code.
20040721	RRC	Made almost all set-valued properties optional and added a rule to default them to the empty set, per agenda item 7 of 2004-07-15 concall.
20040715	RRC	Marked the {message label} property of the Message Reference and Fault Reference components as required.
20040715	RRC	Made the {style} property into a set of xs:anyURI.
20040714	RRC	Added definition of simple types used by the component model (issue 177).
20040713	RRC	Added clarification to interface extensions per issue 220.
20040713	RRC	Added clarification to Binding Operation section (issue 227).
20040713	RRC	Fixed references to Interface Fault components in the Fault Reference component section.
20040713	RRC	Added description of pseudo-schema syntax.
20040714	SW	Made f&p allowed in the remaining places and updated composition rules
20040713	SW	Added negative conformance criteria: not required to process XML1.1 etc.
20040713	SW	Corrected reference to frag ID syntax to for issue 209
20040713	SW	Implemented Jonathan's proposal for issue 160.
20040713	SW	Put ednote in contentious areas asking for extra feedback.
20040712	RRC	Marked all component model properties as REQUIRED or OPTIONAL (issue 213).

20040712	RRC	Added definition for equivalence of list-typed values.
20040712	RRC	Clarified RPC style rules for one-way operations (issue 215).
20040708	JJM	Finished adding clarifications for non-XML type system extensibility.
20040708	JJM	Include the definition of "actual value" from XML Schema (Issue 219).
20040708	JJM	Added resolution to issue 218 (2004Jun/0276.html, including Mark's amendment).
20040708	JJM	Component equivalence (2004Jun/0195.html, 2004Jun/0199.html and ref to the charmod [Issue 210]).
20040706	RRC	Added clarifications for non-XML type system extensibility.
20040706	RRC	Expanded component model definition.
20040706	RRC	Added clarification to section 2.1.1 per resolution of issue 222.
20040706	RRC	Made it possible to use rpc style with schema languages other than XML Schema.
20040702	SW	Made operation/@style be a list of URIs.
20040702	SW	Had forgotten to map to the {type} property of binding.
20040625	SW	Allowed F&P *nearly* everywhere. Sigh.
20040618	SW	Changed F&P composition model to nearest enclosing scope.
20040618	SW	Incorporated Jacek's purpose of bindings text as appropriate.
20040526	SW	Added @address to /definitions/service/endpoint per F2F decision
20040526	SW	Added @type to /definitions/binding per F2F decision
20040519	SW	Renamed wsoap12: to wsoap:.
20040323	JJM	Commented out the (missing) property example.
20040322	RRC	Added definition of wsdli:wsdlLocation attribute.
20040322	JJM	Added faults to properties and features.
20040319	JJM	Use lowercase "should" in notes.
20040319	JJM	Comment out features at service level. Uniformize scope between features and properties.
20040318	JJM	Moved normative notes into the main body of the document.
20040318	JJM	Incorporated the property text from Glen.
20040318	JJM	Addressed comments from Yuxiao Zhao.
20040318	JJM	Updated the feature description, as per Glen and David Booth's suggestions.

20040317	RRC	Removed redundant {styleDefault} property of the interface component.
20040317	JJM	Include comments from Kevin.
20040315	RRC	Added clarification on embedded XML schemas that refer to siblings.
20040315	RRC	Updated RPC signature extension to use #in/#out/#inout/#return tokens.
20040315	RRC	Added explanatory text to types and modularization sections per resolution of issue #102.
20040315	SW	Change binding/{fault,operation}/@name to @ref
20040312	RRC	Fixed appendix D to take the removal of wsdl:message into account.
20040312	RRC	Added definition of wrpc:signature extension attribute.
20040311	SW	Change fault stuff per decision to make faults first class in interfaces.
20040308	SW	Renamed {message} property to {element} and @message to @element
20040305	SW	Added {safety} property
20040227	MJG	Merged in branch Issue143 containing resolution of issue 143
20040227	SW	Dropped {type definitions} property from definitions; leftover from <message> days.
20040226	SW	Working thru various edtodo items.
20040106	JS	Per 18 Dec 2003 telecon decision, added text re: circular includes.
20031204	JS	Per 4 Dec 2003 telecon decision, removed redundant binding/operation/{infault, outfault}/@messageReference.
20031105	JS	Added point to attributes task force recommendation accepted by the working group.
20031104	JS	Mapping to component model for {message} of Fault Reference component indicated that `message` *attribute information item* was optional, but the pseudo syntax and XML representation indicated it was required. Made uniformly optional to allow other type systems as was previously done for {message} of Message Reference component.
20031104	JS	Renamed interface /operation /{input,output} /@body to ./@message and interface /operation /{infault,outfault} /@details to ./@message per 4 Nov face-to-face decision.
20031104	JS	Made interface /operation /{input,output,infault,outfault} /@messageReference optional per 4 Nov face-to-face decision.
20031104	JS	Removed interface/operation/{input,output}/@header per 4 Nov face-to-face decision.

20031102	SW	Updated fault reference components to indicate that if operation's MEP uses MTF then the fault is in the opposite direction as the referenced message and if it use FRM then its in the same direction. Per 10/30 telecon decision.
20031102	SW	Updated operation styles terminology per message #57 of Oct. and the RPC style rules per message #58 of Oct. per decision on 10/30 telecon to consider those status quo.
20031102	SW	Clarified wording in operation styles discussion to better explain the use of the {style} attribute.
20031102	SW	Clarified wording in XML <-> component model mapping section for message reference components to say that {body} and {headers} may not have a value.
20031102	SW	Made interface/operation/(input|output)/@messageReference REQUIRED per 10/30 telecon decision.
20031028	SW	Renamed to wsdl20.xml and updated contents.
20031028	SW	Updated bindings.
20031025	SW	Updated faults.
20031013	JJM	Moved appendix C to a separate document, as per 24 Sep 2003 meeting in Palo Alto, CA.
20031003	SW	Softened <documentation> wording to allow machine processable documentation.
20031002	SW	Changed binding/operation/@name to QName per edtodo.
20030930	SW	Added placeholders for set-attr/get-attr operation styles.
20030929	SW	Inserted Glen Daniels' feature text.
20030919	RRC	Removed import facility for chameleon schemas and added a description of a workaround.
20030918	JJM	Changed message pattern to message exchange pattern, as per WG resolution on 18 Sep. 2003
20030916	RRC	Added editorial note for the missing RPC encoding style.
20030915	RRC	Yet more updates for REQUIRED, OPTIONAL; updated section 3 to reflect the removal of "wsdl:message".
20030911	RRC	More updates for REQUIRED, OPTIONAL; removed diff markup; fixed example C.4.
20030911	RRC	Renamed message reference "name" attribute and property to "messageReference"; fixed incorrect reference to "fault" element in the binding operation section.
20030910	SW	Fixed message references and added proper use of REQUIRED etc. for the part I've gone through so far.

20030910	SW	Updating spec; fixed up interface operation component more.
20030808	JCS	Fixed errors found by IBM\Arthur.
20030804	JCS	Removed Message component per 30 July-1 Aug meeting.
20030803	JCS	Replaced substitution groups with xs:any namespace='##other' per 3 July, 17 July, and 24 July telecons.
20030801	JCS	Made binding/@interface optional per 31 July meeting.
20030724	JCS	Remove @targetResource per 17 July 2003 telecon.
20030612	JJM	Incorporate revised targetResource definition, as per 12 June 2003 telcon.
20030606	JJM	Refer to the two graphics by ID. Indicate pseudo-schemas are not normative.
20030604	JJM	Fixed figures so they don't appear as tables. Fixed markup so it validates.
20030603	JCS	Plugged in jmarsh auto-generated schema outlines
20030529	MJG	Fixed various issues with the XmlRep portions of the spec
20030527	MJG	Added text to **2.2.1 The Interface Component** [p.14] and **2.2.3 Mapping Interface's XML Representation to Component Properties** [p.18] indicating that recursive interface extension is not allowed.
20030523	JJM	Added pseudo-syntax to all but Type and Modularizing sections.
20030523	JJM	Added the "interface" and "targetResource" attribute on <service>.
20030523	JJM	Fixed miscellaneous typos (semi-colon instead of colon, space after parenthesis, etc.).
20030523	JJM	Rewrote the service-resource text and merge it with the introduction.
20030522	JCS	s/set of parts/list of parts/.
20030514	JJM	Updated the service-resource figure, and split the diagram into two.
20030512	JJM	Added service-resource drawing and description.
20030512	JJM	Added syntax summary for the Interface component.
20030428	MJG	Various edits to **3. Types** [p.68] , **E. Examples of Specifications of Extension Elements for Alternative Schema Language Support.** [p.93] to accommodate other type systems and spell out how extensibility elements/attributes play out in such scenarios.
20030428	MJG	Added text to **1.2 Notational Conventions** [p.7] regarding normative nature of schema and validity of WSDL documents

<table>
<tr><td>20030411</td><td>JJM</td><td>Allowed features and properties at the interface, interface operation, binding and binding operation levels, as agreed at the Boston f2f http://lists.w3.org/Archives/Public/www-ws-desc/2003Mar/0019.html.</td></tr>
<tr><td>20030411</td><td>JJM</td><td>Incorporate features and properties’ text from separate document and merged change logs</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Changed title to include ’part 1’</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Changed port to endpoint</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Changed type to interface in binding</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Changed mep to pattern and message exchange pattern to message pattern</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Added text to D.2 PortTypes [p.93]</td></tr>
<tr><td>20030313</td><td>MJG</td><td>Changed portType to interface</td></tr>
<tr><td>20030407</td><td>JJM</td><td>Refined and corrected the definitions for features and properties.</td></tr>
<tr><td>20030304</td><td>JJM</td><td>Filled in blank description of Feature and Property component.</td></tr>
<tr><td>20030303</td><td>MJG</td><td>Skeleton Feature and Property components</td></tr>
<tr><td>20030305</td><td>MJG</td><td>Merged ComponentModelForMEPs branch (1.46.2.5) into main branch (1.54). Below is change log from the branch:
<table>
<tr><th>Date</th><th>Author</th><th>Description</th></tr>
<tr><td>20030220</td><td>MJG</td><td>Minor wording change at suggestion of JJM</td></tr>
<tr><td>20030212</td><td>MJG</td><td>Updated component model to include Fault Reference component. Associated changes to Port Type Operation component</td></tr>
<tr><td>20030211</td><td>MJG</td><td>Changes to component model to support MEPs</td></tr>
</table></td></tr>
<tr><td>20030228</td><td>MJG</td><td>Updated 4.2 Importing Descriptions [p.75] to be consistent in layout with other XML rep sections. Detailed that documentation and extensibility attributes are allowed, per schema</td></tr>
<tr><td>20030228</td><td>MJG</td><td>Updated 4.1 Including Descriptions [p.73] to be consistent in layout with other XML rep sections. Detailed that documentation and extensibility attributes are allowed, per schema</td></tr>
<tr><td>20030228</td><td>MJG</td><td>Updated 2.9.2 XML Representation of Binding Component [p.48] to list type attribute</td></tr>
<tr><td>20030217</td><td>MJG</td><td>Minor edits to wording in 2.4.1 The Interface Operation Component [p.22]</td></tr>
<tr><td>20030213</td><td>MJG</td><td>Added xlink nsdecl to spec element</td></tr>
</table>

<table>
<tr><td>20030213</td><td>MJG</td><td>Incorporated text from dbooths proposal on semantics, per decision 20021031</td></tr>
<tr><td>20030213</td><td>MJG</td><td>Merged operationnames branch (1.37.2.3) into main branch (1.46). Below is the change log from the branch.
<table>
<tr><th>Date</th><th>Author</th><th>Description</th></tr>
<tr><td>20030130</td><td>MJG</td><td>Updated binding section to match changes to port type section WRT operation names</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Added best practice note on operation names and target namespaces to 2.4.1 The Interface Operation Component [p.22]</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Started work on making operations have unique names</td></tr>
</table></td></tr>
<tr><td>20030213</td><td>MJG</td><td>Change name of {message exchange pattern} back to {variety} to consolidate changes due to MEP proposal</td></tr>
<tr><td>20030206</td><td>MJG</td><td>Updated Appendix A to refer to Appendix C</td></tr>
<tr><td>20030204</td><td>MJG</td><td>Tidied up appendix C</td></tr>
<tr><td>20030203</td><td>MJG</td><td>Incorporated resolution to R120</td></tr>
<tr><td>20030124</td><td>MJG</td><td>Fixed error in 2.5.2 XML Representation of Message Reference Component [p.32] which had name attribute information item on input, output and fault element information item being mandatory. Made it optional.</td></tr>
<tr><td>20030123</td><td>JJM</td><td>Change name of {variety} property to {message exchange pattern}</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Updated binding section to match changes to port type section WRT operation names</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Added best practice note on operation names and target namespaces to 2.4.1 The Interface Operation Component [p.22]</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Started work on making operations have unique names</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Added some <emph>, <el>, <att>, &AII;, &EII;, <el> markup</td></tr>
<tr><td>20030120</td><td>MJG</td><td>Incorporated Relax NG section from Amy’s types proposal</td></tr>
<tr><td>20030120</td><td>MJG</td><td>Incorporated DTD section from Amy’s types proposal</td></tr>
<tr><td>2003020</td><td>MJG</td><td>Incorporated Amy’s types proposal except annexes</td></tr>
<tr><td>20030118</td><td>MJG</td><td>Made some changes related to extensibility</td></tr>
<tr><td>20030118</td><td>MJG</td><td>Amended content model for operation to disallow fault element children in the input-only and output-only cases</td></tr>
</table>

20030118	MJG	Removed {extension} properties from Binding components and Port components. Added text relating to how extension elements are expected to annotate the component model.
20030117	MJG	Made further edits related to extensibility model now using substitution groups
20030117	MJG	Added initial draft of section on QName resolution
20030117	MJG	Reworked section on extensibility
20030116	MJG	Added text regarding multiple operations with the same {name} in a single port type
20030116	MJG	Added section on symbol spaces
20030116	MJG	Removed various ednotes
20030116	MJG	Added section on component equivalence
20030116	MJG	More work on include and import
20021201	MJG	Did some work on wsdl:include
20021127	MJG	Added placeholder for wsdl:include
20021127	MJG	Cleaned up language concerning `targetNamespace` *attribute information item* **2.1.2.1 targetNamespace attribute information item** [p.13]
20021127	MJG	changed the language regarding extensibility elements in **2.1.2 XML Representation of Definitions Component** [p.12] .
20021127	MJG	Moved all issues into issues document (../issues/wsd-issues.xml)
20021127	MJG	Removed name attribute from definitions element
20021127	MJG	Removed ’pseudo-schema’
20021121	JJM	Updated media type draft appendix ednote to match minutes.
20021111	SW	Added appendix to record migration issues.
20021107	JJM	Incorporated and started adapting SOAP’s media type draft appendix.
20021010	MJG	Added port type extensions, removed service type.
20020910	MJG	Removed parameterOrder from spec, as decided at September 2002 FTF
20020908	MJG	Updated parameterOrder description, fixed some spelling errors and other types. Added ednote to discussion of message parts
20020715	MJG	AM Rewrite
20020627	JJM	Changed a few remaining <emph> to either <att> or <el>, depending on context.

20020627	SW	Converted portType stuff to be infoset based and improved doc structure more.
20020627	SW	Converted message stuff to be infoset based and improved doc structure more.
20020625	SW	Mods to take into account JJM comments.
20020624	JJM	Fixed spec so markup validates.
20020624	JJM	Upgraded the stylesheet and DTD
20020624	JJM	Added sections for references and change log.
20020624	JJM	Removed Jeffrey from authors :-(Added Gudge :-)
20020620	SW	Started adding abstract model
20020406	SW	Created document from WSDL 1.1