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

W3C Recommendation 26 June 2007

This version:
    http://www.w3.org/TR/2007/REC-wsdl20-20070626
Latest version:
    http://www.w3.org/TR/wsdl20
Previous version:
    http://www.w3.org/TR/2007/PR-wsdl20-20070523
Editors:
    Roberto Chinnici, Sun Microsystems
    Jean-Jacques Moreau, Canon
    Arthur Ryman, IBM
    Sanjiva Weerawarana, WSO2

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: XHTML with Z Notation, PDF, PostScript, XML, and plain text.

See also **translations**.

Abstract

This document describes the Web Services Description Language Version 2.0 (WSDL 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 the conformance criteria for documents in 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 the W3C Recommendation of Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language for review by W3C Members and other interested parties. It has been produced by the Web Services Description Working Group, which is part of the W3C Web Services Activity.

Please send comments about this document to the public public-ws-desc-comments@w3.org mailing list (public archive).

The Working Group released a test suite along with an implementation report. A diff-marked version against the previous version of this document is available.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document is governed by the 24 January 2002 CPP as amended by the W3C Patent Policy Transition Procedure. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

Appendices

---

1. Introduction

Web Services Description Language Version 2.0 (WSDL 2.0) provides a model and an XML format for describing Web services. WSDL 2.0 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 the conformance criteria for documents in this language.

The companion specification, *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* [*WSDL 2.0 Adjuncts [p.79]*] describes extensions for message exchange patterns, operation safety, operation styles and binding extensions (for SOAP [*SOAP 1.2 Part 1: Messaging Framework (Second Edition) [p.80]*] and HTTP [*IETF RFC 2616 [p.79]*]).

1.1 Service Description

WSDL 2.0 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 to separate independent design concerns.

At an abstract level, WSDL 2.0 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 The Meaning of a Service Description

A WSDL 2.0 service description indicates how potential clients are intended to interact with the described service. It represents an assertion that the described service fully implements and conforms to what the WSDL 2.0 document describes. For example, as further explained in section **6.1.1 Mandatory extensions** [p.72] , if the WSDL 2.0 document specifies a particular optional extension, the functionality implied by that extension is only optional to the client. It must be supported by the Web service.

A WSDL 2.0 interface describes potential interactions with a Web service, not required interactions. The declaration of an operation in a WSDL 2.0 interface is not an assertion that the interaction described by the operation must occur. Rather it is an assertion that if such an interaction is (somehow) initiated, then the declared operation describes how that interaction is intended to occur.

1.3 Document Conformance

An *element information item* (as defined in [*XML Information Set [p.78]*]) whose namespace name is "http://www.w3.org/ns/wsdl" and whose local part is `description` conforms to this specification if it is valid according to the XML Schema for that element as defined by this specification (http://www.w3.org/2007/06/wsdl/wsdl20.xsd) and additionally adheres to all the constraints contained in this specification and conforms to the specifications of any extensions contained in it. Such a conformant *element information item* constitutes a *WSDL 2.0 document*.

The definition of the WSDL 2.0 language is based on the XML Information Set [*XML Information Set [p.78]*] but also imposes many semantic constraints over and above structural conformance to this XML Infoset. In order to precisely describe these constraints, and as an aid in precisely defining the meaning of each WSDL 2.0 document, the WSDL 2.0 specification defines a component model **2. Component Model** [p.11] as an additional layer of abstraction above the XML Infoset. Constraints and meaning are defined in terms of this component model, and the definition of each component includes a mapping that specifies how values in the component model are derived from corresponding items in the XML Infoset.

An XML 1.0 document that is valid with respect to the WSDL 2.0 XML Schema and that maps to a valid WSDL 2.0 Component Model is conformant to the WSDL 2.0 specification.

1.4 Notational Conventions

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

1.4.1 RFC 2119 Keywords

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.78]*].

1.4.2 RFC 3986 Namespaces

Namespace names of the general form:

- "http://example.org/..." and
- "http://example.com/..."

represent application or context-dependent URIs [*IETF RFC 3986 [p.78]*].

1.4.3 XML Schema anyURI

This specification uses the XML Schema type `xs:anyURI` (see [*XML Schema: Datatypes [p.79]*]). It is defined so that `xs:anyURI` values are essentially IRIs (see [*IETF RFC 3987 [p.78]*]). The conversion from `xs:anyURI` values to an actual URI is via an escaping procedure defined by (see [*XLink 1.0 [p.78]*]), which is identical in most respects to IRI Section 3.1 (see [*IETF RFC 3987 [p.78]*]).

For interoperability, WSDL authors are advised to avoid the US-ASCII characters: "<", ">", ’"’, space, "{", "}", "|", "\", "^", and "‘", which are allowed by the `xs:anyURI` type, but disallowed in IRIs.

1.4.4 Prefixes and Namespaces Used in This Specification

This specification uses predefined namespace prefixes throughout; they are given in the following list. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [*XML Namespaces [p.78]*]).

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

Prefix	Namespace	Notes
wsdl	"http://www.w3.org/ns/wsdl"	Defined by this specification.
wsdli	"http://www.w3.org/ns/wsdl-instance"	Defined by this specification **7.1 wsdli:wsdlLocation attribute information item** [p.75] .
wsdlx	"http://www.w3.org/ns/wsdl-extensions"	Defined by this specification **3.3 Describing Messages that Refer to Services and Endpoints** [p.65] .
wrpc	"http://www.w3.org/ns/wsdl/rpc"	Defined by WSDL 2.0: Adjuncts [*WSDL 2.0 Adjuncts [p.79]*].
wsoap	"http://www.w3.org/ns/wsdl/soap"	Defined by WSDL 2.0: Adjuncts [*WSDL 2.0 Adjuncts [p.79]*].
whttp	"http://www.w3.org/ns/wsdl/http"	Defined by WSDL 2.0: Adjuncts [*WSDL 2.0 Adjuncts [p.79]*].
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [*XML Schema: Structures [p.78]*], [*XML Schema: Datatypes [p.79]*].
xsi	"http://www.w3.org/2001/XMLSchema-instance"	Defined in the W3C XML Schema specification [*XML Schema: Structures [p.78]*], [*XML Schema: Datatypes [p.79]*].

1.4.5 Terms Used in This Specification

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.78]*], the expression "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, an IRI-reference, etc.

Inlined Schema

An XML schema that is defined in the `wsdl:types` *element information item* of a WSDL 2.0 description. For example, an XML Schema defined in an `xs:schema` *element information item* **3.1.2 Inlining XML Schema** [p.63] .

1.4.6 XML Information Set Properties

This specification refers to properties in the XML Information Set [*XML Information Set [p.78]*]. Such properties are denoted by square brackets, e.g. [children], [attributes].

1.4.7 WSDL 2.0 Component Model Properties

This specification defines and refers to properties in the WSDL 2.0 Component Model **2. Component Model** [p.11] . Such properties are denoted by curly brackets, e.g. {name [p.17] }, {interfaces [p.13] }.

This specification uses a consistent naming convention for component model properties that refer to components. If a property refers to a required or optional component, then the property name is the same as the component name. If a property refers to a set of components, then the property name is the pluralized form of the component name.

1.4.8 Z Notation

Z Notation [*Z Notation Reference Manual [p.80]*] was used in the development of this specification. Z Notation is a formal specification language that is based on standard mathematical notation. The Z Notation for this specification has been verified using the Fuzz 2000 type-checker [*Fuzz 2000 [p.80]*].

Since Z Notation is not widely known, it is not included the normative version of this specification. However, it is included in a non-normative version which allows to dynamically hide and show the Z Notation. Browsers correctly display the mathematical Unicode characters, provided that the required fonts are installed. Mathematical fonts for Mozilla Firefox can be downloaded from the Mozilla Web site.

The Z Notation was used to improve the quality of the normative text that defines the Component Model, and to help ensure that the test suite covered all important rules implied by the Component Model. However, the Z Notation is non-normative, so any conflict between it and the normative text is an error in the Z Notation. Readers and implementers may nevertheless find the Z Notation useful in cases where the normative text is unclear.

There are two elements of Z Notation syntax that conflict with the notational conventions described in the preceding sections. In Z Notation, square brackets are used to introduce basic sets, e.g. [ID], which conflicts with the use of square brackets to denote XML Information Set properties **1.4.6 XML Information Set Properties** [p.10] . Also, in Z Notation, curly brackets are used to denote set display and set comprehension, e.g. {1, 2, 3}, which conflicts with the use of curly brackets to denote WSDL 2.0 Component Model properties **1.4.7 WSDL 2.0 Component Model Properties** [p.10] . However, the intended meaning of square and curly brackets should be clear from their context and this minor notational conflict should not cause any confusion.

1.4.9 BNF Pseudo-Schemas

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, and "|" represents choice. Attributes are conventionally assigned a value which corresponds to their type, as defined in the normative schema. Elements with simple content are conventionally assigned a value which corresponds to the type of their content, as defined in the normative schema. Pseudo schemas do not include extension points for brevity.

```
<!-- 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>
```

1.4.10 Assertions

Assertions about WSDL 2.0 documents and components that are not enforced by the normative XML schema for WSDL 2.0 are marked by a dagger symbol (†) at the end of a sentence. Each assertion has been assigned a unique identifier that consists of a descriptive textual prefix and a unique numeric suffix. The numeric suffixes are assigned sequentially and never reused so there may be gaps in the sequence. The assertion identifiers MAY be used by implementations of this specification for any purpose, e.g. error reporting.

The assertions and their identifiers are summarized in section **E. Assertion Summary** [p.93] .

2. Component Model

This section describes the conceptual model of WSDL 2.0 as a set of components with attached properties, which collectively describe a Web service. This model is called the *Component Model* of WSDL 2.0. A *valid WSDL 2.0 component model* is a set of WSDL 2.0 components and properties that satisfy all the requirements given in this specification as indicated by keywords whose interpretation is defined by RFC 2119 [*IETF RFC 2119 [p.78]*].

Components are typed collections of properties that correspond to different aspects of Web services. Each subsection herein describes a different type of component, its defined properties, and its representation as an XML Infoset [*XML Information Set [p.78]*].

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 serializable in XML 1.0 format but are independent of any particular serialization of the component model. Component definitions use a subset (see **2.14 XML Schema 1.0 Simple Types Used in the Component Model** [p.57]) of the simple types defined by the XML Schema 1.0 specification [*XML Schema: Datatypes [p.79]*].

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 2.0 descriptions** [p.67] .

A component model can be extracted from a given XML Infoset which conforms to the XML Schema for WSDL 2.0 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 2.0 descriptions** [p.67] .

This document does not specify a means of producing an XML Infoset representation from a component model instance. In particular, there are in general many valid ways to modularize a given component model instance into one or more XML Infosets.

2.1 Description

2.1.1 The Description Component

At a high level, the Description [p.13] component is just a container for two categories of components: WSDL 2.0 components and type system components.

WSDL 2.0 components are interfaces, bindings and services. Type system components are element declarations and type definitions.

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.78]*], 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 2.0; see **6. Language Extensibility** [p.72] . In the case where they define information equivalent to that of a XML Schema global element declaration, they can be treated as if they were such a declaration.

This specification does not define the behavior of a WSDL 2.0 document that uses multiple schema languages for describing type system components simultaneously.

An Element Declaration component defines the name and content model of an *element information item* such as that defined by an XML Schema global element declaration. It has a {name} property that is the QName of the *element information item* and a {system} property that is the namespace IRI of the extension *element information item*s for the type system, e.g. the namespace of XML Schema.

A Type Definition component defines the content model of an *element information item* such as that defined by an XML Schema global type definition. It has a {name} property that is the QName of the type and a {system} property that is the namespace IRI of the extension *element information item*s for the type system, e.g. the namespace of XML Schema.

Interface [p.17] , Binding [p.39] , Service [p.52] , Element Declaration [p.12] , and Type Definition [p.12] components are directly contained in the Description [p.13] component and are referred to as *top-level components*. The top-level WSDL 2.0 components contain other components, e.g. Interface Operation [p.25] and Endpoint [p.55] , which are referred to as *nested components*. Nested components may contain other nested components. The component that contains a nested component is referred to as the *parent* of the nested component. Nested components have a {parent} property that is a reference to their parent component.

The properties of the Description component are as follows:

- {interfaces} OPTIONAL. A set of Interface [p.17] components.
- {bindings} OPTIONAL. A set of Binding [p.39] components.
- {services} OPTIONAL. A set of Service [p.52] components.
- {element declarations} OPTIONAL. A set of Element Declaration [p.12] components.
- {type definitions} REQUIRED. A set of Type Definition [p.12] components.

The set of top-level components contained in the Description [p.13] component associated with an initial WSDL 2.0 document consists of the components defined in the initial document, plus the components associated with the WSDL 2.0 documents that the initial document includes, plus the components defined by other WSDL 2.0 documents in the namespaces that the initial document imports. The component model makes no distinction between the components that are defined in the initial document versus those that are defined in the included documents or imported namespaces. However, any WSDL 2.0 document that contains component definitions that refer by QName to WSDL 2.0 components that belong to a different namespace MUST contain a `wsdl:import` *element information item* for that namespace (see **4.2 Importing Descriptions** [p.68]). Furthermore, all QName references, whether to the same or to different namespaces must resolve to components (see **2.17 QName resolution** [p.59]).

When using the XML Schema language to describe type system components, the inclusion of Element Declaration [p.12] components and Type Definition [p.12] components in a Description [p.13] component is governed by the rules in **3.1 Using W3C XML Schema Definition Language** [p.61] .

In addition to WSDL 2.0 components and type system components, additional extension components MAY be added via extensibility **6. Language Extensibility** [p.72] . Further, additional properties to WSDL 2.0 and type system components MAY also be added via extensibility.

2.1.2 XML Representation of Description Component

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

WSDL 2.0 descriptions are represented in XML by one or more WSDL 2.0 Information Sets (Infosets), that is one or more `description` *element information items*. A WSDL 2.0 Infoset contains representations for a collection of WSDL 2.0 components that share a common target namespace and zero or more `wsdl:import` *element information items* **4.2 Importing Descriptions** [p.68] that correspond to a collection with components from multiple target namespaces.

The components directly defined or included within a Description [p.13] 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 value of the `targetNamespace` *attribute information item* SHOULD be dereferencable.† [p.99] It SHOULD resolve to a human or machine processable document that directly or indirectly defines the intended semantics of those components.† [p.99] It MAY resolve to a WSDL 2.0 document that provides service description information for that namespace.† [p.100]

If a WSDL 2.0 document is split into multiple WSDL 2.0 documents (which may be combined as needed via **4.1 Including Descriptions** [p.67]), then the `targetNamespace` *attribute information item* SHOULD resolve to a master WSDL 2.0 document that includes all the WSDL 2.0 documents needed for that service description.† [p.94] This approach enables the WSDL 2.0 component designator fragment identifiers to be properly resolved.

Components that belong to imported namespaces have different target namespace values than that of the importing WSDL 2.0 document. Thus importing is the mechanism to use components from one namespace in the definition of components from another namespace.

Note that each WSDL 2.0 document or type system component of the same kind must be uniquely identified by its qualified name. That is, if two distinct components of the same kind (Interface [p.17] , Binding [p.39] , etc.) are in the same target namespace, then their QNames MUST be unique. However, different kinds of components (e.g., an Interface [p.17] component and a Binding [p.39] component) MAY have the same QName. Thus, QNames of components must be unique within the space of those components in a given target namespace.

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

- A [local name] of `description`.
- A [namespace name] of "http://www.w3.org/ns/wsdl".
- One or more *attribute information items* amongst its [attributes] as follows:
    - A REQUIRED `targetNamespace` *attribute information item* as described below in **2.1.2.1 targetNamespace attribute information item** [p.15] .
    - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information items* amongst its [children], in order as follows:† [p.94]

1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
2. Zero or more *element information items* from among the following, in any order:
   - Zero or more `include` *element information items* (see **4.1 Including Descriptions** [p.67])
   - Zero or more `import` *element information items* (see **4.2 Importing Descriptions** [p.68])
   - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
3. An OPTIONAL `types` *element information item* (see **3. Types** [p.59]).
4. Zero or more *element information items* from among the following, in any order:
   - `interface` *element information items* (see **2.2.2 XML Representation of Interface Component** [p.18]).
   - `binding` *element information items* (see **2.7.2 XML Representation of Binding Component** [p.39]).
   - `service` *element information items* (see **2.12.2 XML Representation of Service Component** [p.52]).
   - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/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 `description` *element information item*. Interface [p.17] , Binding [p.39] and Service [p.52] 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*. Its value MUST be an absolute IRI (see [*IETF RFC 3987 [p.78]*]) and should be dereferencable. † [p.94]

2.1.3 Mapping Description's XML Representation to Component Properties

The mapping from the XML Representation of the `description` *element information item* (see **2.1.2 XML Representation of Description Component** [p.13]) to the properties of the Description [p.13] component is described in Table 2-1 [p.16] .

Table 2-1. Mapping from XML Representation to Description Component Properties

Property	Value
{interfaces [p.13] }	The set of Interface [p.17] components corresponding to all the `interface` *element information items* in the [children] of the `description` *element information item*, if any, plus any included (via `wsdl:include`) or imported (via `wsdl:import`) Interface [p.17] components (see **4. Modularizing WSDL 2.0 descriptions** [p.67]).
{bindings [p.13] }	The set of Binding [p.39] components corresponding to all the `binding` *element information items* in the [children] of the `description` *element information item*, if any, plus any included (via `wsdl:include`) or imported (via `wsdl:import`) Binding [p.39] components (see **4. Modularizing WSDL 2.0 descriptions** [p.67]).
{services [p.13] }	The set of Service [p.52] components corresponding to all the `service` *element information items* in the [children] of the `description` *element information item*, if any, plus any included (via `wsdl:include`) or imported (via `wsdl:import`) Service [p.52] components (see **4. Modularizing WSDL 2.0 descriptions** [p.67]).
{element declarations [p.13] }	The set of Element Declaration [p.12] components corresponding to all the element declarations defined as descendants of the `types` *element information item*, if any, plus any included (via `xs:include`) or imported (via `xs:import`) Element Declaration [p.12] components. At a minimum this will include all the global element declarations defined by XML Schema `element` *element information items*. 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*. Each XML Schema element declaration MUST have a unique QName.[† [p.103]]

{type definitions [p.13] }	The set of Type Definition [p.12] components corresponding to all the type definitions defined as descendants of the `types` *element information item*, if any, plus any included (via `xs:include`) or imported (via `xs:import`) Type Definition [p.12] components. In addition, the built-in datatypes defined by XML Schema Part 2: Datatypes Second Edition [*XML Schema: Datatypes [p.79]*], namely the nineteen primitive datatypes `xs:string`, `xs:boolean`, `xs:decimal`, `xs:float`, `xs:double`, `xs:duration`, `xs:dateTime`, `xs:time`, `xs:date`, `xs:gYearMonth`, `xs:gYear`, `xs:gMonthDay`, `xs:gDay`, `xs:gMonth`, `xs:hexBinary`, `xs:base64Binary`, `xs:anyURI`, `xs:QName`, `xs:NOTATION`, and the twenty-five derived datatypes `xs:normalizedString`, `xs:token`, `xs:language`, `xs:NMTOKEN`, `xs:NMTOKENS`, `xs:Name`, `xs:NCName`, `xs:ID`, `xs:IDREF`, `xs:IDREFS`, `xs:ENTITY`, `xs:ENTITIES`, `xs:integer`, `xs:nonPositiveInteger`, `xs:negativeInteger`, `xs:long`, `xs:int`, `xs:short`, `xs:byte`, `xs:nonNegativeInteger`, `xs:unsignedLong`, `xs:unsignedInt`, `xs:unsignedShort`, `xs:unsignedByte`, `xs:positiveInteger`. The set MAY also include any definitions from some other type system which describes the [attributes] and [children] properties of an *element information item*. Each XML Schema type definition MUST have a unique QName. † [p.103]

2.2 Interface

2.2.1 The Interface Component

An Interface [p.17] 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.

An interface can optionally extend one or more other interfaces. To avoid circular definitions, an interface MUST NOT appear in the set of interfaces it extends, either directly or indirectly. † [p.101] The set of operations available in an interface includes all the operations defined by the interfaces it extends directly or indirectly, together with any operations it directly defines. The operations directly defined on an interface are referred to as the *declared* operations of the interface. In the process, operation components that are equivalent per **2.15 Equivalence of Components** [p.58] are treated as one single component. The interface extension mechanism behaves in a similar way for all other components that can be defined inside an interface, namely Interface Fault [p.21] components.

Interfaces are named constructs and can be referred to by QName (see **2.17 QName resolution** [p.59]). For instance, Binding [p.39] components refer to interfaces in this way.

The properties of the Interface component are as follows:

- {name} REQUIRED. An *xs:QName*.

- {extended interfaces} OPTIONAL. A set of declared Interface [p.17] components which this interface extends.
- {interface faults} OPTIONAL. The set of declared Interface Fault [p.21] components. Note that the namespace name of the {name [p.21] } property of each Interface Fault [p.21] in this set is the same as the namespace name of the {name [p.17] } property of this Interface [p.17] component.
- {interface operations} OPTIONAL. A set of declared Interface Operation [p.25] components. Note that the namespace name of the {name [p.25] } property of each Interface Operation [p.25] in this set is the same as the namespace name of the {name [p.17] } property of this Interface [p.17] component.

For each Interface [p.17] component in the {interfaces [p.13] } property of a Description [p.13] component, the {name [p.17] } property MUST be unique. † [p.101]

2.2.2 XML Representation of Interface Component

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

The XML representation for an Interface [p.17] component is an *element information item* with the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- One or more *attribute information items* 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 element]** [p.19] .
  - An OPTIONAL `extends` *attribute information item* as described below in **2.2.2.2 extends attribute information item** [p.19] .
  - An OPTIONAL `styleDefault` *attribute information item* as described below in **2.2.2.3 styleDefault attribute information item** [p.19] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information items* amongst its [children], in order, as follows:

1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
2. Zero or more *element information items* from among the following, in any order:
   - Zero or more `fault` *element information items* **2.3.2 XML Representation of Interface Fault Component** [p.22] .
   - Zero or more `operation` *element information items* **2.4.2 XML Representation of Interface Operation Component** [p.27] .
   - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

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

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the [parent] `description` *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 whitespace-separated list of *xs:QName*.

The list of *xs:QName* in an `extends` *attribute information item* MUST NOT contain duplicates. † [p.101]

2.2.2.3 `styleDefault` *attribute information item*

The `styleDefault` *attribute information item* indicates the default style (see **2.4.1.2 Operation Style** [p.27]) used to construct the {element declaration [p.31] } properties of {interface message references [p.25] } of all operations contained within the [owner element] `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*. Its value, if present, MUST contain absolute IRIs (see [*IETF RFC 3987 [p.78]*]). † [p.95]

2.2.3 Mapping Interface's XML Representation to Component Properties

The mapping from the XML Representation of the `interface` *element information item* (see **2.2.2 XML Representation of Interface Component** [p.18]) to the properties of the Interface [p.17] component is as described in Table 2-2 [p.20] .

Table 2-2. Mapping from XML Representation to Interface Component Properties

Property	Value
{name [p.17] }	The QName whose local name is actual value of the `name` *attribute information item* and whose namespace name is the actual value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item*
{extended interfaces [p.18] }	The set of Interface [p.17] components resolved to by the values in the `extends` *attribute information item*, if any (see **2.17 QName resolution** [p.59]).
{interface faults [p.18] }	The set of Interface Fault [p.21] components corresponding to the `fault` *element information items* in [children], if any.
{interface operations [p.18] }	The set of Interface Operation [p.25] components corresponding to the `operation` *element information items* in [children], if any.

Recall that, per **2.2.1 The Interface Component** [p.17] , the Interface [p.17] components in the {extended interfaces [p.18] } property of a given Interface [p.17] component MUST NOT contain that Interface [p.17] component in any of their {extended interfaces [p.18] } 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 [p.21] component describes a fault that MAY occur during invocation of an operation of the interface. The Interface Fault [p.21] 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 [p.25] component.

The Interface Fault [p.21] 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.

Faults other than the ones described in the Interface [p.17] component may also be generated at run-time, i.e. faults are an open set. The Interface [p.17] component describes faults that have application level semantics, i.e. that the client or service is expected to handle, and potentially recover from, as part of the application processing logic. For example, an Interface [p.17] component that accepts a credit card number may describe faults that indicate the credit card number is invalid, has been reported stolen, or has expired. The Interface [p.17] component does not describe general system faults such as network failures, out of memory conditions, out of disk space conditions, invalid message formats, etc., although these faults may be generated as part of the message exchange. Such general system faults can reasonably be expected to occur in any message exchange and explicitly describing them in an Interface [p.17] component is therefore uninformative.

The properties of the Interface Fault component are as follows:

- {name} REQUIRED. An *xs:QName*.
- {message content model} REQUIRED. An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.† [p.101] A value of *#any* indicates that the fault content is any single element. A value of *#none* indicates there is no fault content. A value of *#other* indicates that the fault content is described by some other extension property that references a declaration in a non-XML extension type system. A value of *#element* indicates that the fault consists of a single element described by the global element declaration referenced by the {element declaration [p.21] } property. This property is used only when the fault is described using an XML-based data model.
- {element declaration} OPTIONAL. A reference to an Element Declaration [p.12] component in the {element declarations [p.13] } property of the Description [p.13] component. This element represents the content or “payload” of the fault. When the {message content model [p.21] } property has the value *#any* or *#none* the {element declaration [p.21] } property MUST be empty.† [p.101]
- {parent} REQUIRED. The Interface [p.17] component that contains this component in its {interface faults [p.18] } property.

For each Interface Fault [p.21] component in the {interface faults [p.18] } property of an Interface [p.17] component, the {name [p.21] } property must be unique. Note that this constraint is enforced by the normative WSDL 2.0 XML schema.

Interface Fault [p.21] components are uniquely identified by the QName of the enclosing Interface [p.17] component and QName of the Interface Fault [p.21] component itself.

Note:

Despite having a {name [p.21] } property, Interface Fault [p.21] components cannot be identified solely by their QName. Indeed, two Interface [p.17] components whose {name [p.17] } property value has the same namespace name, but different local names, can contain Interface Fault [p.21] components with the same {name [p.21] } property value. Thus, the {name [p.21] } property of Interface Fault [p.21] component is not sufficient to form the unique identity of an Interface Fault [p.21] component. A method for uniquely identifying components is defined in **A.2 Fragment Identifiers** [p.82] . See **A.2.5 The Interface Fault Component** [p.84] for the definition of the fragment identifier for the Interface Fault [p.21] component.

In cases where, due to an interface extending one or more other interfaces, two or more Interface Fault [p.21] components have the same value for their {name [p.21] } property, then the component models of those Interface Fault [p.21] components MUST be equivalent (see **2.15 Equivalence of Components** [p.58]). † [p.101] If the Interface Fault [p.21] components are equivalent then they are considered to collapse into a single component. Within the same Interface [p.17] component, if two Interface Fault [p.21] components are not equivalent then their {name [p.21] } properties MUST NOT be equal.

Note that, due to the above rules, if two interfaces that have the same value for the namespace name of their {name [p.17] } property also have one or more faults that have the same value for their {name [p.21] } property, then those two interfaces cannot both form part of the derivation chain of a derived interface unless those faults are the same fault.

For the above reason, it is considered good practice to ensure, where necessary, that the local name of the {name [p.21] } property of Interface Fault [p.21] components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error. † [p.102]

If a type system NOT based on the XML Infoset [*XML Information Set [p.78]*] is in use (as considered in **3.2 Using Other Schema Languages** [p.65]) then additional properties would need to be added to the Interface Fault [p.21] component (along with extension attributes to its XML representation) to allow associating such message types with the message reference.

2.3.2 XML Representation of Interface Fault Component

```
<description>
  <interface>
    <fault
          name="xs:NCName"
          element="union of xs:QName, xs:token"? >
      <documentation />*
    </fault>
  </interface>
</description>
```

The XML representation for an Interface Fault [p.21] component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- One 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 element]** [p.23] .
  - An OPTIONAL `element` *attribute information item* as described below in **2.3.2.2 element attribute information item with fault [owner element]** [p.23] .
  - Zero or more namespace qualified *attribute information item*s whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information item*s (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information item* s whose [namespace name] is NOT " http://www.w3.org/ns/wsdl " .

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

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 element]

The `element` *attribute information item* refers, by QName, to an Element Declaration [p.12] 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 a union of *xs:QName* and *xs:token* where the allowed token values are *#any*, *#none*, or *#other*.

2.3.3 Mapping Interface Fault's XML Representation to Component Properties

The mapping from the XML Representation of the `fault` *element information item* (see **2.3.2 XML Representation of Interface Fault Component** [p.22]) to the properties of the Interface Fault [p.21] component is as described in Table 2-3 [p.24] .

Table 2-3. Mapping from XML Representation to Interface Fault Component Properties

Property	Value
{name [p.21] }	The QName whose local name is the actual value of the `name` *attribute information item*. and whose namespace name is the actual value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item* of the [parent] `interface` *element information item*.
{message content model [p.21] }	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 *#other*.
{element declaration [p.21] }	If the `element` *attribute information item* is present and its value is a QName, then the Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component resolved to by the value of the `element` *attribute information item* (see **2.17 QName resolution** [p.59]); otherwise empty. If the `element` *attribute information item* has a value, then it MUST resolve to an Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component. † [p.95]
{parent [p.21] }	The Interface [p.17] component corresponding to the `interface` *element information item* in [parent].

2.4 Interface Operation

2.4.1 The Interface Operation Component

An Interface Operation [p.25] component describes an operation that a given interface supports. An operation is an interaction with the service consisting of a set of (ordinary and fault) messages exchanged between the service and the other parties involved in the interaction. 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 [p.25] } 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 {interface message references [p.25] } and {interface fault references [p.25] } 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 Adjuncts [p.79]*] defines a set of such patterns and defines identifying IRIs any of which MAY be used as the value of the {message exchange pattern [p.25] } property.

The properties of the Interface Operation component are as follows:

- {name} REQUIRED. An *xs:QName*.
- {message exchange pattern} REQUIRED. An *xs:anyURI* identifying the message exchange pattern used by the operation. This *xs:anyURI* MUST be an absolute IRI (see [*IETF RFC 3987 [p.78]*]). † [p.102]
- {interface message references} OPTIONAL. A set of Interface Message Reference [p.30] components for the ordinary messages the operation accepts or sends.
- {interface fault references} OPTIONAL. A set of Interface Fault Reference [p.35] components for the fault messages the operation accepts or sends.
- {style} OPTIONAL. A set of *xs:anyURI*s identifying the rules that were used to construct the {element declaration [p.31] } properties of {interface message references [p.25] }. (See **2.4.1.2 Operation Style** [p.27] .) These *xs:anyURI*s MUST be absolute IRIs (see [*IETF RFC 3986 [p.78]*]). † [p.102]
- {parent} REQUIRED. The Interface [p.17] component that contains this component in its {interface operations [p.18] } property.

For each Interface Operation [p.25] component in the {interface operations [p.18] } property of an Interface [p.17] component, the {name [p.25] } property MUST be unique. Note that this constraint is enforced by the normative WSDL 2.0 XML schema.

Interface Operation [p.25] components are uniquely identified by the QName of the enclosing Interface [p.17] component and QName of the Interface Operation [p.25] component itself.

Note:

Despite having a {name [p.25] } property, Interface Operation [p.25] components cannot be identified solely by their QName. Indeed, two Interface [p.17] components whose {name [p.17] } property value has the same namespace name, but different local names, can contain Interface Operation [p.25] components with the same {name [p.25] } property value. Thus, the {name [p.25] } property of Interface Operation [p.25] components is not sufficient to form the unique identity of an Interface Operation [p.25] component. A method for uniquely identifying components is defined in **A.2 Fragment Identifiers** [p.82] . See **A.2.6 The Interface Operation Component** [p.84] for the definition of the fragment identifier for the Interface Operation [p.25] component.

In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation [p.25] components have the same value for their {name [p.25] } property, then the component models of those Interface Operation components MUST be equivalent (see **2.15 Equivalence of Components** [p.58]). † [p.102] If the Interface Operation [p.25] components are equivalent then they are considered to collapse into a single component. Within the same Interface [p.17] component, if two Interface Operation [p.25] components are not equivalent then their {name [p.25] } properties MUST NOT be equal.

Note that, due to the above rules, if two interfaces that have the same value for the namespace name of their {name [p.17] } property also have one or more operations that have the same value for their {name [p.25] } property, then those two interfaces cannot both form part of the derivation chain of a derived interface unless those operations are the same operation.

For the above reason, it is considered good practice to ensure, where necessary, that the {name [p.25] } property of Interface Operation [p.25] components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error. † [p.103]

More than one Interface Fault Reference [p.35] component in the {interface fault references [p.25] } property of an Interface Operation [p.25] component may refer to the same message label. In that case, the listed fault types define alternative fault messages. This allows one to indicate that there is more than one type of fault that is related to that message.

2.4.1.1 Message Exchange Pattern

This section describes some aspects of message exchange patterns in more detail. Refer to the *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* specification [*WSDL 2.0 Adjuncts [p.79]*] for a complete discussion of the semantics of message exchange patterns in general, as well as the definitions of the message exchange patterns that are predefined by WSDL 2.0.

A *placeholder message* is a template for an actual message as described by an Interface Message Reference [p.30] component. Although a placeholder message is not itself a component, it is useful to regard it as having both a {message label [p.30] } and a {direction [p.31] } property which define the values of the actual Interface Message Reference [p.30] component that corresponds to it. A placeholder message is also associated with some node that exchanges the message with the service. Furthermore, a placeholder message may be designated as optional in the exchange.

A *fault propagation ruleset* specifies the relation between the Interface Fault Reference [p.35] and Interface Message Reference [p.30] components of an Interface Operation [p.25] component. The *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* specification [*WSDL 2.0 Adjuncts [p.79]*] defines three fault propagation rulesets which we will refer to as *fault-replaces-message*, *message-triggers-fault*, and *no-faults*. These three fault propagation rulesets are used by the predefined message exchange patterns defined in [*WSDL 2.0 Adjuncts [p.79]*]. Other message exchange patterns can define additional fault propagation rulesets.

A *message exchange pattern* is a template for the exchange of one or more messages, and their associated faults, between the service and one or more other nodes as described by an Interface Operation [p.25] component. The service and the other nodes are referred to as the *participants* in the exchange. More specifically, a message exchange pattern consists of a sequence of one or more placeholder messages. Each placeholder message within this sequence is uniquely identified by its {message label [p.30] } property. A message exchange pattern is itself uniquely identified by an absolute IRI, which is used as the value of the {message exchange pattern [p.25] } property of the Interface Operation [p.25] component, and which specifies the fault propagation ruleset that its faults obey. † [p.103]

2.4.1.2 Operation Style

An operation style specifies additional information about an operation. For example, an operation style may define structural constraints on the element declarations of the interface message reference or interface fault components used by the operation. This additional information in no way affects the messages and faults exchanged with the service and it can therefore be safely ignored in that context. However, the additional information can be used for other purposes, for example, improved code generation. The {style [p.25] } property of the Interface Operation [p.25] component contains a set of zero or more IRIs that identify operation styles. An Interface Operation [p.25] component MUST satisfy the specification defined by each operation style identified by its {style [p.25] } property. † [p.103] If no Interface Operation [p.25] component can simultaneously satisfy all of the styles, the document is invalid.

If the {style [p.25] } property of an Interface Operation [p.25] component does have a value, then that value (a set of IRIs) specifies the rules that were used to define the element declarations (or other properties that define the message and fault contents; see **3.2 Using Other Schema Languages** [p.65]) of the Interface Message Reference [p.30] or Interface Fault [p.21] components used by the operation. Although a given operation style has the ability to constrain *all* input and output messages and faults of an operation, it MAY choose to constrain any combination thereof, e.g. only the messages, or only the inputs.

Please refer to the *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* specification [*WSDL 2.0 Adjuncts [p.79]*] for particular operation style definitions.

2.4.2 XML Representation of Interface Operation Component

```
<description>
  <interface>
    <operation
          name="xs:NCName"
          pattern="xs:anyURI"?
          style="list of xs:anyURI"? >
      <documentation />*
      [ <input /> | <output /> | <infault /> | <outfault /> ]*
    </operation>
  </interface>
</description>
```

The XML representation for an Interface Operation [p.25] component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.4.2.1 name attribute information item with operation [owner element]** [p.28] .

- An OPTIONAL `pattern` *attribute information item* as described below in **2.4.2.2 pattern attribute information item with operation [owner element]** [p.29] .
- An OPTIONAL `style` *attribute information item* as described below in **2.4.2.3 style attribute information item with operation [owner element]** [p.29] .
- Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

- One or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. One or more *element information items* from among the following, in any order:
     - One or more *element information items* from among the following, in any order:
       - Zero or more `input` *element information items* (see **2.5.2 XML Representation of Interface Message Reference Component** [p.31]).
       - Zero or more `output` *element information items* (see **2.5.2 XML Representation of Interface Message Reference Component** [p.31]).
       - Zero or more `infault` *element information items* (see **2.6.2 XML Representation of Interface Fault Reference** [p.35]).
       - Zero or more `outfault` *element information items* (see **2.6.2 XML Representation of Interface Fault Reference** [p.35]).
     - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

2.4.2.1 `name` *attribute information item* with `operation` [owner element]

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*.

2.4.2.2 `pattern` *attribute information item* with `operation` [owner element]

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*. Note that its value must be an absolute IRI (see [*IETF RFC 3987 [p.78]*]).

2.4.2.3 `style` *attribute information item* with `operation` [owner element]

The `style` *attribute information item* indicates the rules that were used to construct the {element declaration [p.31] } properties of the Interface Message Reference [p.30] components which are members of the {interface message references [p.25] } property of the [owner element] 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*. Note that its value must be an absolute IRI (see [*IETF RFC 3987 [p.78]*]).

2.4.3 Mapping Interface Operation's XML Representation to Component Properties

The mapping from the XML Representation of the `operation` *element information item* (see **2.4.2 XML Representation of Interface Operation Component** [p.27]) to the properties of the Interface Operation component (see **2.4.1 The Interface Operation Component** [p.24]) is as described in Table 2-4 [p.29] .

Table 2-4. Mapping from XML Representation to Interface Operation Component Properties

Property	Value
{name [p.25] }	The QName whose local name is the actual value of the `name` *attribute information item* and whose namespace name is the actual value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item* of the [parent] `interface` *element information item*.
{message exchange pattern [p.25] }	The actual value of the `pattern` *attribute information item*; otherwise 'http://www.w3.org/ns/wsdl/in-out'.
{interface message references [p.25] }	The set of message references corresponding to the `input` and `output` *element information items* in [children], if any.
{interface fault references [p.25] }	The set of interface fault references corresponding to the `infault` and `outfault` *element information items* in [children], if any.
{style [p.25] }	The set containing the IRIs in the actual value of the `style` *attribute information item*, if present; otherwise the set containing the IRIs in the actual value of the `styleDefault` *attribute information item* of the [parent] `interface` *element information item*, if present; otherwise empty.
{parent [p.25] }	The Interface [p.17] component corresponding to the `interface` *element information item* in [parent].

2.5 Interface Message Reference

2.5.1 The Interface Message Reference Component

An Interface Message Reference [p.30] component defines the content, or *payload*, of a message exchanged in an operation. By default, the message content is defined by an XML-based type system such as XML Schema. Other type systems may be used via the WSDL 2.0 type system extension mechanism.

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 an Interface Message Reference [p.30] component is to associate an actual message element (XML element declaration or some other declaration (see **3.2 Using Other Schema Languages** [p.65])) 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 element assignment made by the Interface Message Reference [p.30] component.

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

- {message label} REQUIRED. An *xs:NCName*. This property identifies the role this message plays in the {message exchange pattern [p.25] } of the Interface Operation [p.25] component this message is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern. † [p.103]

- {direction} REQUIRED. An *xs:token* with one of the values *in* or *out*, indicating whether the message is coming to the service or going from the service, respectively.† [p.102] The direction MUST be the same as the direction of the message identified by the {message label [p.30] } property in the {message exchange pattern [p.25] } of the Interface Operation [p.25] component this is contained within.† [p.102]

- {message content model} REQUIRED. An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.† [p.102] 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 *#other* indicates that the message content is described by some other extension property that references a declaration in a non-XML extension type system. A value of *#element* indicates that the message consists of a single element described by the global element declaration referenced by the {element declaration [p.31] } property. This property is used only when the message is described using an XML-based data model.

- {element declaration} OPTIONAL. A reference to an Element Declaration [p.12] component in the {element declarations [p.13] } property of the Description [p.13] component. This element represents the content or "payload" of the message. When the {message content model [p.31] } property has the value *#any* or *#none*, the {element declaration [p.31] } property MUST be empty.† [p.102]

- {parent} REQUIRED. The Interface Operation [p.25] component that contains this component in its {interface message references [p.25] } property.

For each Interface Message Reference [p.30] component in the {interface message references [p.25] } property of an Interface Operation [p.25] component, its {message label [p.30] } property MUST be unique.† [p.102]

If a type system not based upon the XML Infoset is in use (as considered in **3.2 Using Other Schema Languages** [p.65]), then additional properties would need to be added to the Interface Message Reference [p.30] component (along with extension attributes to its XML representation) to allow associating such message types with the message reference.

2.5.2 XML Representation of Interface Message Reference Component

```
<description>
  <interface>
    <operation>
      <input
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:token"? >
        <documentation />*
      </input>
      <output
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:token"? >
        <documentation />*
      </output>
    </operation>
  </interface>
</description>
```

The XML representation for an Interface Message Reference [p.30] 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/ns/wsdl"
- Zero or more *attribute information items* 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 element]** [p.32] .
  - An OPTIONAL `element` *attribute information item* as described below in **2.5.2.2 element attribute information item with input or output [owner element]** [p.32] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

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

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 element]

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*, *#none*, or *#other*.

2.5.3 Mapping Interface Message Reference's XML Representation to Component Properties

The mapping from the XML Representation of the interface message reference *element information item* (see **2.5.2 XML Representation of Interface Message Reference Component** [p.31]) to the properties of the Interface Message Reference [p.30] component (see **2.5.1 The Interface Message Reference Component** [p.30]) is as described in Table 2-5 [p.33] and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the {message exchange pattern [p.25] } of the parent Interface Operation [p.25] component.

Define the *message direction* of the *element information item* to be *in* if its local name is `input`, and *out* if its local name is `output`.

Note that the `messageLabel` *attribute information item* of an interface message reference *element information item* must be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction.

If the `messageLabel` *attribute information item* of an interface message reference *element information item* is present, then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction. † [p.95]

If the `messageLabel` *attribute information item* of an interface message reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction. † [p.95]

Define the *effective message label* of an interface message reference *element information item* to be either the actual value of the `messageLabel` *attribute information item* if it is present, or the {message label [p.30] } of the unique placeholder message with {direction [p.31] } equal to the message direction if the *attribute information item* is absent.

If the local name is `input` then the message exchange pattern MUST have at least one placeholder message with direction "In". † [p.95]

If the local name is `output` then the message exchange pattern MUST have at least one placeholder message with direction "Out". † [p.95]

If the local name is `infault` then the message exchange pattern MUST support at least one fault in the "In" direction. † [p.95]

If the local name is `outfault` then the message exchange pattern MUST support at least one fault in the "Out" direction. † [p.95]

Table 2-5. Mapping from XML Representation to Interface Message Reference Component Properties

Property	Value
{message label [p.30] }	The effective message label.
{direction [p.31] }	The message direction.
{message content model [p.31] }	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 *#other*.
{element declaration [p.31] }	If the `element` *attribute information item* is present and its value is a QName, then the Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component resolved to by the value of the `element` *attribute information item* (see **2.17 QName resolution** [p.59]); otherwise empty. If the `element` *attribute information item* has a value, then it MUST resolve to an Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component. † [p.95]
{parent [p.31] }	The Interface Operation [p.25] component corresponding to the `interface` *element information item* in [parent].

2.6 Interface Fault Reference

2.6.1 The Interface Fault Reference Component

An Interface Fault Reference [p.35] component associates a defined type, specified by an Interface Fault [p.21] 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 an Interface Fault Reference [p.35] component is to associate an actual message type (XML element declaration or some other declaration (see **3.2 Using Other Schema Languages** [p.65]) for message content, as specified by an Interface Fault [p.21] component) with a fault message occurring in the pattern. In order to identify the fault message it describes, the Interface Fault Reference [p.35] component uses the message label of the message the fault is associated with, as a key.

As indicated earlier, the companion specification [*WSDL 2.0 Adjuncts [p.79]*] defines several *fault propagation rulesets* that a given message exchange pattern may use. For the ruleset *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 ruleset *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 message exchange pattern.

The properties of the Interface Fault Reference component are as follows:

- {interface fault} REQUIRED. An Interface Fault [p.21] component in the {interface faults [p.18] } property of the [parent] Interface Operation [p.25] component's [parent] Interface [p.17] component, or an Interface [p.17] component that it directly or indirectly extends. Identifying the Interface Fault [p.21] component therefore indirectly defines the actual content or payload of the fault message.

- {message label} REQUIRED. An *xs:NCName*. This property identifies the message this fault relates to among those defined in the {message exchange pattern [p.25] } property of the Interface Operation [p.25] component it is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern. † [p.102]

- {direction} REQUIRED. A *xs: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 propagation ruleset used in the message exchange pattern of the operation. † [p.102] For example, if the ruleset *fault-replaces-message* is used, then a fault that refers to an outgoing message would have a {direction [p.35] } property value of *out*. On the other hand, if the ruleset *message-triggers-fault* is used, then a fault that refers to an outgoing message would have a {direction [p.35] } property value of *in* as the fault travels in the opposite direction of the message.

- {parent} REQUIRED. The Interface Operation [p.25] component that contains this component in its {interface fault references [p.25] } property.

For each Interface Fault Reference [p.35] component in the {interface fault references [p.25] } property of an Interface Operation [p.25] component, the combination of its {interface fault [p.35] } and {message label [p.35] } properties MUST be unique. † [p.102]

2.6.2 XML Representation of Interface Fault Reference

```
<description>
  <interface>
    <operation>
      <infault
            ref="xs:QName"
            messageLabel="xs:NCName"? >
        <documentation />*
      </infault>*
      <outfault
            ref="xs:QName"
            messageLabel="xs:NCName"? >
        <documentation />*
      </outfault>*
    </operation>
  </interface>
</description>
```

The XML representation for an Interface Fault Reference [p.35] 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/ns/wsdl"
- One or more *attribute information items* 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 element]** [p.36] .
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.6.2.2 messageLabel attribute information item with infault, or outfault [owner element]** [p.36] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

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

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 `ref` *attribute information item* is *xs:QName*.

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

The `messageLabel` *attribute information item* identifies the message in the message exchange pattern of the given `operation` *element information item* that is associated with this fault.

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*.

The `messageLabel` *attribute information item* MUST be present in the XML representation of an Interface Fault Reference [p.35] component with a given {direction [p.35] }, if the {message exchange pattern [p.25] } of the parent Interface Operation [p.25] component has more than one fault with that direction. † [p.95] Recall that the *fault propagation ruleset* of the {message exchange pattern [p.25] } specifies the relation between faults and messages. For example, the *fault-replaces-message* ruleset specifies that the faults have the same direction as the messages, while the *message-triggers-fault* ruleset specifies that the faults have the opposite direction from the messages.

2.6.3 Mapping Interface Fault Reference's XML Representation to Component Properties

The mapping from the XML Representation of the message reference *element information item* (see **2.6.2 XML Representation of Interface Fault Reference** [p.35]) to the properties of the Interface Fault Reference component (see **2.6.1 The Interface Fault Reference Component** [p.34]) is as described in Table 2-6 [p.37] and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the {message exchange pattern [p.25] } of the parent Interface Operation [p.25] component.

Define the *fault direction* of the *element information item* to be *in* if its local name is `infault` and *out* if its local name is `outfault`.

Define the *message direction* of the *element information item* to be the {direction [p.31] } of the placeholder message associated with the fault as specified by the fault propagation ruleset of the message exchange pattern.

The `messageLabel` *attribute information item* of an interface fault reference *element information item* MUST be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

If the `messageLabel` *attribute information item* of an interface fault reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

If the `messageLabel` *attribute information item* of an interface fault reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

Define the *effective message label* of an interface fault reference *element information item* to be either the actual value of the `messageLabel` *attribute information item* if it is present, or the {message label [p.30] } of the unique placeholder message whose {direction [p.31] } is equal to the message direction if the *attribute information item* is absent.

Table 2-6. Mapping from XML Representation to Interface Fault Reference Component Properties

Property	Value
{interface fault [p.35] }	The Interface Fault [p.21] component from {interface faults [p.18] } property of the parent Interface [p.17] component, or an Interface [p.17] component that it directly or indirectly extends, with {name [p.21] } equal to the actual value of the `ref` *attribute information item*.
{message label [p.35] }	The effective message label.
{direction [p.35] }	The fault direction.
{parent [p.35] }	The Interface Operation [p.25] component corresponding to the `interface` *element information item* in [parent].

2.7 Binding

2.7.1 The Binding Component

A Binding [p.39] component describes a concrete message format and transmission protocol which may be used to define an endpoint (see **2.13 Endpoint** [p.54]). That is, a Binding [p.39] component defines the implementation details necessary to access the service.

Binding [p.39] components can be used to describe such information in a reusable manner for any interface or specifically for a given interface. Furthermore, binding information MAY be specified on a per-operation basis (see **2.9.1 The Binding Operation Component** [p.44]) within an interface, in addition to across all operations of an interface.

If a Binding [p.39] component specifies any operation-specific binding details (by including Binding Operation [p.44] components) or any fault binding details (by including Binding Fault [p.42] components), then it MUST specify an interface the Binding [p.39] component applies to, so as to indicate which interface the operations come from. † [p.98]

Conversely, a Binding [p.39] component which omits any operation-specific binding details and any fault binding details MAY omit specifying an interface. Binding [p.39] components that do not specify an interface MAY be used to specify operation-independent binding details for Service [p.52] components with different interfaces. That is, such Binding [p.39] components are reusable across one or more interfaces.

No concrete binding details are given in this specification. The companion specification, *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* [*WSDL 2.0 Adjuncts [p.79]*] defines such bindings for SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework (Second Edition) [p.80]*] and HTTP [*IETF RFC 2616 [p.79]*]. Other specifications MAY define additional binding details. Such specifications are expected to annotate the Binding [p.39] component (and its sub-components) with additional properties and specify the mapping from the XML representation to these properties.

A Binding [p.39] component that defines bindings for an Interface [p.17] component MUST define bindings for all the operations of that Interface [p.17] component.† [p.98] The bindings can occur via defaulting rules which allow one to specify default bindings for all operations and faults (see, for example [*WSDL 2.0 Adjuncts [p.79]*]) or by defining bindings for each Interface Operation [p.25] and Interface Fault [p.21] component of the Interface [p.17] component.

Similarly, whenever a reusable Binding [p.39] component (i.e. one that does not specify an Interface [p.17] component) is applied to a specific Interface [p.17] component in the context of an Endpoint [p.55] component (see **2.13.1 The Endpoint Component** [p.54]), the Binding [p.39] component MUST define bindings for each Interface Operation [p.25] and Interface Fault [p.21] component of the Interface [p.17] component, via a combination of properties defined on the Binding [p.39] component itself and default binding rules specific to its binding type.† [p.98]

A Binding [p.39] component that defines bindings for an Interface [p.17] component MUST define bindings for all the faults of that Interface [p.17] component that are referenced from any of the operations in that Interface [p.17] component.† [p.98] As for the case of operations, the binding can be defined by defaulting rules. Note that only the faults actually referenced by operations are required to have bindings.

Bindings are named constructs and can be referred to by QName (see **2.17 QName resolution** [p.59]). For instance, Endpoint [p.55] components refer to bindings in this way.

The properties of the Binding component are as follows:

- {name} REQUIRED. An *xs:QName*.
- {interface} OPTIONAL. An Interface [p.17] component indicating the interface for which binding information is being specified.
- {type} REQUIRED. An *xs:anyURI*. This *xs:anyURI* MUST be an absolute IRI as defined by [*IETF RFC 3987 [p.78]*].† [p.98] The value of this IRI indicates what kind of concrete binding details are contained within this Binding [p.39] component. Specifications (such as [*WSDL 2.0 Adjuncts [p.79]*]) 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.
- {binding faults} OPTIONAL. A set of Binding Fault [p.42] components.
- {binding operations} OPTIONAL. A set of Binding Operation [p.44] components.

For each Binding [p.39] component in the {bindings [p.13] } property of a Description [p.13] component, the {name [p.39] } property MUST be unique.† [p.98]

2.7.2 XML Representation of Binding Component

```
<description>
  <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI" >
```

```
    <documentation />*
    [ <fault /> | <operation /> ]*
  </binding>
</description>
```

The XML representation for a Binding [p.39] component is an *element information item* with the following Infoset properties:

- A [local name] of `binding`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- Two or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.7.2.1 name attribute information item with binding [owner element]** [p.40] .
  - An OPTIONAL `interface` *attribute information item* as described below in **2.7.2.2 interface attribute information item with binding [owner element]** [p.41] .
  - An REQUIRED `type` *attribute information item* as described below in **2.7.2.3 type attribute information item with binding [owner element]** [p.41] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information items* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more *element information items* from among the following, in any order:
     - Zero or more `fault` *element information items* (see **2.8.2 XML Representation of Binding Fault Component** [p.43]).
     - Zero or more `operation` *element information items* (see **2.9.2 XML Representation of Binding Operation Component** [p.45]).
     - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl". Such *element information items* are considered to be binding extension elements(see **2.7.2.4 Binding extension elements** [p.41]).

2.7.2.1 `name` *attribute information item* with `binding` [owner element]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `description` *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*.

2.7.2.2 `interface` *attribute information item* with `binding` [owner element]

The `interface` *attribute information item* refers, by QName, to an Interface [p.17] 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*.

2.7.2.3 `type` *attribute information item* with `binding` [owner element]

The `type` *attribute information item* identifies the kind of binding details contained in the Binding [p.39] 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*.

2.7.2.4 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 [p.39] component with additional properties and specify the mapping from the XML representation to those properties.

2.7.3 Mapping Binding's XML Representation to Component Properties

The mapping from the XML Representation of the `binding` *element information item* (see **2.7.2 XML Representation of Binding Component** [p.39]) to the properties of the Binding [p.39] component (see **2.7.1 The Binding Component** [p.38]) is as described in Table 2-7 [p.41] .

Table 2-7. Mapping from XML Representation to Binding Component Properties

Property	Value
{name [p.39] }	The QName whose local name is the actual value of the `name` *attribute information item* and whose namespace name is the actual value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item*.
{interface [p.39] }	The Interface [p.17] component resolved to by the actual value of the `interface` *attribute information item* (see **2.17 QName resolution** [p.59]), if any.
{type [p.39] }	The actual value of the `type` *attribute information item*.
{binding faults [p.39] }	The set of Binding Fault [p.42] components corresponding to the `fault` *element information items* in [children], if any.
{binding operations [p.39] }	The set of Binding Operation [p.44] components corresponding to the `operation` *element information items* in [children], if any.

2.8 Binding Fault

2.8.1 The Binding Fault Component

A Binding Fault [p.42] 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 its {name [p.21] } property.

Note that the fault does not occur by itself -it occurs as part of a message exchange as defined by an Interface Operation [p.25] component (and its binding counterpart the Binding Operation [p.44] component). Thus, the fault binding information specified in a Binding Fault [p.42] component describes how faults that occur within a message exchange of an operation will be formatted and carried in the transport.

The properties of the Binding Fault component are as follows:

- {interface fault} REQUIRED. An Interface Fault [p.21] component in the {interface faults [p.18] } property of the Interface [p.17] component identified by the {interface [p.39] } property of the parent Binding [p.39] component, or an Interface [p.17] component that that Interface [p.17] component directly or indirectly extends. This is the Interface Fault [p.21] component for which binding information is being specified.
- {parent} REQUIRED. The Binding [p.39] component that contains this component in its {binding faults [p.39] } property.

For each Binding Fault [p.42] component in the {binding faults [p.39] } property of a Binding [p.39] component, the {interface fault [p.42] } property MUST be unique.† [p.98] That is, one cannot define multiple bindings for the same fault within a given Binding [p.39] component.

2.8.2 XML Representation of Binding Fault Component

```
<description>
  <binding>
    <fault
          ref="xs:QName" >
      <documentation />*
    </fault>
  </binding>
</description>
```

The XML representation for a Binding Fault [p.42] component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- One or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.8.2.1 ref attribute information item with fault [owner element]** [p.43] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl". Such *element information items* are considered to be binding fault extension elements as described further below (see **2.8.2.2 Binding Fault extension elements** [p.43]).

2.8.2.1 `ref` *attribute information item* with `fault` [owner element]

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*.

2.8.2.2 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 items* are defined by the specification for those *element information items*. Such specifications are expected to annotate the Binding Fault [p.42] component with additional properties and specify the mapping from the XML representation to those properties.

2.8.3 Mapping Binding Fault's XML Representation to Component Properties

The mapping from the XML Representation of the `fault` *element information item* (see **2.8.2 XML Representation of Binding Fault Component** [p.43]) to the properties of the Binding Fault [p.42] component (see **2.8.1 The Binding Fault Component** [p.42]) is as described in Table 2-8 [p.44] .

Table 2-8. Mapping from XML Representation to Binding Fault Component Properties

Property	Value
{interface fault [p.42] }	The Interface Fault [p.21] component corresponding to the actual value of the `ref` *attribute information item.*
{parent [p.42] }	The Binding [p.39] component corresponding to the `binding` *element information item* in [parent].

2.9 Binding Operation

2.9.1 The Binding Operation Component

The Binding Operation [p.44] 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 its {name [p.25] } property.

The properties of the Binding Operation component are as follows:

- {interface operation} REQUIRED. An Interface Operation [p.25] component in the {interface operations [p.18] } property of the Interface [p.17] component identified by the {interface [p.39] } property of the [parent] Binding [p.39] component, or an Interface component that that Interface [p.17] component directly or indirectly extends. This is the Interface Operation [p.25] component for which binding information is being specified.
- {binding message references} OPTIONAL. A set of Binding Message Reference [p.47] components.
- {binding fault references} OPTIONAL. A set of Binding Fault Reference [p.49] components.
- {parent} REQUIRED. The Binding [p.39] component that contains this component in its {binding operations [p.39] } property.

For each Binding Operation [p.44] component in the {binding operations [p.39] } property of a Binding [p.39] component, the {interface operation [p.44] } property MUST be unique.† [p.99] That is, one cannot define multiple bindings for the same operation within a given Binding [p.39] component.

2.9.2 XML Representation of Binding Operation Component

```
<description>
  <binding>
    <operation
          ref="xs:QName" >
      <documentation />*
      [ <input /> | <output /> | <infault /> | <outfault /> ]*
    </operation>
  </binding>
</description>
```

The XML representation for a Binding Operation [p.44] component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- One or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.9.2.1 ref attribute information item with operation [owner element]** [p.46] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information items* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more *element information items* from among the following, in any order:
     - Zero or more `input` *element information items* (see **2.10 Binding Message Reference** [p.46])
     - Zero or more `output` *element information items* (see **2.10 Binding Message Reference** [p.46])
     - Zero or more `infault` *element information items* (see **2.11 Binding Fault Reference** [p.49])
     - Zero or more `outfault` *element information items* (see **2.11 Binding Fault Reference** [p.49])
     - Zero or more namespace-qualified *element information item* whose [namespace name] is NOT " http://www.w3.org/ns/wsdl ". Such *element information items* are considered to be binding operation extension elements as described below (see **2.9.2.2 Binding Operation extension elements** [p.46]).

2.9.2.1 `ref` *attribute information item* with `operation` [owner element]

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*.

2.9.2.2 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 [p.44] component with additional properties and specify the mapping from the XML representation to those properties.

2.9.3 Mapping Binding Operation's XML Representation to Component Properties

The mapping from the XML Representation of the `operation` *element information item* (see **2.9.2 XML Representation of Binding Operation Component** [p.45]) to the properties of the Binding Operation [p.44] component is as described in Table 2-9 [p.46] .

Table 2-9. Mapping from XML Representation to Binding Operation Component Properties

Property	Value
{interface operation [p.44] }	The Interface Operation [p.25] component corresponding to the actual value of the `ref` *attribute information item*.
{binding message references [p.44] }	The set of Binding Message Reference [p.47] components corresponding to the `input` and `output` *element information item*s in [children], if any.
{binding fault references [p.44] }	The set of Binding Fault Reference [p.49] components corresponding to the `infault` and `outfault` *element information item*s in [children], if any.
{parent [p.44] }	The Binding [p.39] component corresponding to the `binding` *element information item* in [parent].

2.10 Binding Message Reference

2.10.1 The Binding Message Reference Component

A Binding Message Reference [p.47] 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:

- {interface message reference} REQUIRED. An Interface Message Reference [p.30] component among those in the {interface message references [p.25] } property of the Interface Operation [p.25] component being bound by the containing Binding Operation [p.44] component.
- {parent} REQUIRED. The Binding Operation [p.44] component that contains this component in its {binding message references [p.44] } property.

For each Binding Message Reference [p.47] component in the {binding message references [p.44] } property of a Binding Operation [p.44] component, the {interface message reference [p.47] } property MUST be unique. † [p.99] That is, the same message cannot be bound twice within the same operation.

2.10.2 XML Representation of Binding Message Reference Component

```
<description>
  <binding>
    <operation>
      <input
            messageLabel="xs:NCName"? >
        <documentation />*
      </input>
      <output
            messageLabel="xs:NCName"? >
        <documentation />*
      </output>
    </operation>
  </binding>
</description>
```

The XML representation for a Binding Message Reference [p.47] 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/ns/wsdl".
- Zero or more *attribute information items* amongst its [attributes] as follows:
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.10.2.1 messageLabel attribute information item with input or output [owner element]** [p.48] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).

2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl". Such *element information items* are considered to be binding message reference extension elements as described below (see **2.10.2.2 Binding Message Reference extension elements** [p.48]).

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

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*.

2.10.2.2 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 items* are defined by the specification for those *element information items*. Such specifications are expected to annotate the Binding Message Reference [p.47] component with additional properties and specify the mapping from the XML representation to those properties.

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

The mapping from the XML Representation of the `binding` *element information item* (see **2.10.2 XML Representation of Binding Message Reference Component** [p.47]) to the properties of the Binding Message Reference [p.47] component is as described in Table 2-10 [p.49] and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the {message exchange pattern [p.25] } of the Interface Operation [p.25] component being bound.

Define the *message direction* of the *element information item* to be *in* if its local name is `input` and *out* if its local name is `output`.

Note that the `messageLabel` *attribute information item* of a binding message reference *element information item* must be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction.

If the `messageLabel` *attribute information item* of a binding message reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

If the `messageLabel` *attribute information item* of a binding message reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

Define the *effective message label* of a binding message reference *element information item* to be either the actual value of the `messageLabel` *attribute information item* if it is present, or the {message label [p.30] } of the unique placeholder message with {direction [p.31] } equal to the message direction if the *attribute information item* is absent.

Table 2-10. Mapping from XML Representation to Binding Message Reference Component Properties

Property	Value
{interface message reference [p.47] }	The Interface Message Reference [p.30] component in the {interface message references [p.25] } of the Interface Operation [p.25] component being bound with {message label [p.30] } equal to the effective message label.
{parent}	The Binding Operation [p.44] component corresponding to the `operation` *element information item* in [parent].

2.11 Binding Fault Reference

2.11.1 The Binding Fault Reference Component

A Binding Fault Reference [p.49] component describes a concrete binding of a particular fault participating in an operation to a particular concrete message format.

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

- {interface fault reference} REQUIRED. An Interface Fault Reference [p.35] component among those in the {interface fault references [p.25] } property of the Interface Operation [p.25] component being bound by the parent Binding Operation [p.44] component.
- {parent} REQUIRED. The Binding Operation [p.44] component that contains this component in its {binding fault references [p.44] } property.

For each Binding Fault Reference [p.49] component in the {binding fault references [p.44] } property of a Binding Operation [p.44] component, the {interface fault reference [p.49] } property MUST be unique.[†] [p.98] That is, the same fault cannot be bound twice within the same operation.

2.11.2 XML Representation of Binding Fault Reference Component

```
<description>
  <binding>
    <operation>
      <infault
            ref="xs:QName"
            messageLabel="xs:NCName"?>
        <documentation />*
      </infault>
      <outfault
            ref="xs:QName"
            messageLabel="xs:NCName"?>
        <documentation />*
```

```
      </outfault>
    </operation>
  </binding>
</description>
```

The XML representation for a Binding Fault Reference [p.49] 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/ns/wsdl".
- One or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.11.2.1 ref attribute information item with infault or outfault [owner element]** [p.50] .

    An OPTIONAL `messageLabel` *attribute information item* as described below in **2.11.2.2 messageLabel attribute information item with infault or outfault [owner element]** [p.50] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl". Such *element information items* are considered to be binding fault reference extension elements as described below (see **2.11.2.3 Binding Fault Reference extension elements** [p.51]).

2.11.2.1 `ref` *attribute information item* with `infault` or `outfault` [owner element]

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

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*.

2.11.2.3 Binding Fault Reference extension elements

Binding Fault Reference extension elements are used to provide information specific to a particular fault 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 Fault Reference [p.49] component with additional properties and specify the mapping from the XML representation to those properties.

2.11.3 Mapping Binding Fault Reference's XML Representation to Component Properties

The mapping from the XML Representation of the `binding` *element information item* (see **2.11.2 XML Representation of Binding Fault Reference Component** [p.49]) to the properties of the Binding Fault Reference [p.49] component is as described in Table 2-11 [p.52] and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the {message exchange pattern [p.25] } of the Interface Operation [p.25] component being bound.

Define the *fault direction* of the *element information item* to be *in* if its local name is `infault` and *out* if its local name is `outfault`.

Define the *message direction* of the *element information item* to be the {direction [p.31] } of the placeholder message associated with the fault as specified by the fault propagation ruleset of the message exchange pattern.

The `messageLabel` *attribute information item* of a binding fault reference *element information item* MUST be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

If the `messageLabel` *attribute information item* of a binding fault reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

If the `messageLabel` *attribute information item* of a binding fault reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction. † [p.96]

Define the *effective message label* of a binding fault reference *element information item* to be either the actual value of the `messageLabel` *attribute information item* if it is present, or the {message label [p.30] } of the unique placeholder message with {direction [p.31] } equal to the message direction if the *attribute information item* is absent.

There MUST be an Interface Fault Reference [p.35] component in the {interface fault references [p.25] } of the Interface Operation [p.25] being bound with {message label [p.35] } equal to the effective message label and with {interface fault [p.35] } equal to an Interface Fault [p.21] component with {name [p.21] } equal to the actual value of the `ref` *attribute information item*.† [p.99]

Table 2-11. Mapping from XML Representation to Binding Fault Reference Component Properties

Property	Value
{interface fault reference [p.49] }	The Interface Fault Reference [p.35] component in the {interface fault references [p.25] } of the Interface Operation [p.25] being bound with {message label [p.35] } equal to the effective message label, and with {interface fault [p.35] } equal to an Interface Fault [p.21] component with {name [p.21] } equal to the actual value of the `ref` *attribute information item*.
{parent}	The Binding Operation [p.44] component corresponding to the `operation` *element information item* in [parent].

2.12 Service

2.12.1 The Service Component

A Service [p.52] component describes a set of endpoints (see **2.13 Endpoint** [p.54]) 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 **2.17 QName resolution** [p.59]).

The properties of the Service component are as follows:

- {name} REQUIRED. An *xs:QName*.
- {interface} REQUIRED. An Interface [p.17] component.
- {endpoints} REQUIRED. A non-empty set of Endpoint [p.55] components.

For each Service [p.52] component in the {services [p.13] } property of a Description [p.13] component, the {name [p.52] } property MUST be unique.† [p.103]

2.12.2 XML Representation of Service Component

```
<description>
  <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />*
    <endpoint />+
  </service>
</description>
```

The XML representation for a Service [p.52] component is an *element information item* with the following Infoset properties:

- A [local name] of `service`
- A [namespace name] of "http://www.w3.org/ns/wsdl"
- Two or more *attribute information items* amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.12.2.1 name attribute information item with service [owner element]** [p.53] .
  - A REQUIRED `interface` *attribute information item* as described below in **2.12.2.2 interface attribute information item with service [owner element]** [p.53] .
  - Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".
- One or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. One or more *element information items* from among the following, in any order:
     - One or more `endpoint` *element information items* (see **2.13.2 XML Representation of Endpoint Component** [p.55]
     - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

2.12.2.1 `name` *attribute information item* with `service` [owner element]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `description` *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*.

2.12.2.2 `interface` *attribute information item* with `service` [owner element]

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.*.

2.12.3 Mapping Service's XML Representation to Component Properties

The mapping from the XML Representation of the `service` *element information item* (see **2.12.2 XML Representation of Service Component** [p.52]) to the properties of the Service [p.52] component is as described in Table 2-12 [p.54] .

Table 2-12. Mapping from XML Representation to Service Component Properties

Property	Value
{name [p.52] }	The QName whose local name is the actual value of the `name` *attribute information item*, and whose namespace name is the actual value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item*.
{interface [p.52] }	The Interface [p.17] component resolved to by the actual value of the `interface` *attribute information item* (see **2.17 QName resolution** [p.59]).
{endpoints [p.52] }	The Endpoint [p.55] components corresponding to the `endpoint` *element information item*s in [children].

2.13 Endpoint

2.13.1 The Endpoint Component

An Endpoint [p.55] component defines the particulars of a specific endpoint at which a given service is available.

Endpoint [p.55] components are local to a given Service [p.52] component (see **A.2 Fragment Identifiers** [p.82]).

The Binding [p.39] component specified by the {binding [p.55] } property of an Endpoint [p.55] component is said to be *applied* to the Interface [p.17] component which is the value of the {interface [p.52] } property of the parent Service [p.52] component of the Endpoint [p.55] . According to the constraints given below, if this Binding [p.39] component has an {interface [p.39] } property, its value must be the Interface [p.17] component the Binding [p.39] component is applied to.

The {address [p.55] } property is optional to allow for means other than IRIs to be used, e.g. a WS-Addressing Endpoint Reference [*WSA 1.0 Core [p.79]*]. It is also possible that, in certain scenarios, an address will not be required, in which case this property may be absent.

The properties of the Endpoint component are as follows:

- {name} REQUIRED. An *xs:NCName*.
- {binding} REQUIRED. A Binding [p.39] component.
- {address} OPTIONAL. An *xs:anyURI*. This *xs:anyURI* MUST be an absolute IRI as defined by [*IETF RFC 3987 [p.78]*]. † [p.100] If present, the value of this attribute represents the network address at which the service indicated by the parent Service [p.52] component's {interface [p.52] } property is offered via the binding referred to by the {binding [p.55] } property. **Note** that the presence in this property of the characters "?" and "#" can conflict with those potentially added by the query string serialization mechanism, as defined in Serialization as "application/x-www-form-urlencoded" ([*WSDL 2.0 Adjuncts [p.79]*], section 6.8.2).
- {parent} REQUIRED. The Service [p.52] component that contains this component in its {endpoints [p.52] } property.

For each Endpoint [p.55] component in the {endpoints [p.52] } property of a Service [p.52] component, the {name [p.55] } property MUST be unique. Note that this constraint is enforced by the normative WSDL 2.0 XML schema.

For each Endpoint [p.55] component in the {endpoints [p.52] } property of a Service [p.52] component, the {binding [p.55] } property MUST either be a Binding [p.39] component with an unspecified {interface [p.39] } property or a Binding [p.39] component with an {interface [p.39] } property equal to the {interface [p.52] } property of the Service [p.52] component. † [p.100]

2.13.2 XML Representation of Endpoint Component

```
<description>
  <service>
    <endpoint
          name="xs:NCName"
          binding="xs:QName"
          address="xs:anyURI"? >
      <documentation />*
    </endpoint>+
  </service>
</description>
```

The XML representation for a Endpoint [p.55] component is an *element information item* with the following Infoset properties:

- A [local name] of `endpoint`.
- A [namespace name] of "http://www.w3.org/ns/wsdl".
- Two or more *attribute information items* amongst its [attributes] as follows:

- A REQUIRED `name` *attribute information item* as described below in **2.13.2.1 name attribute information item with endpoint [owner element]** [p.56] .
- A REQUIRED `binding` *attribute information item* as described below in **2.13.2.2 binding attribute information item with endpoint [owner element]** [p.56] .
- An OPTIONAL `address` *attribute information item* as described below in **2.13.2.3 address attribute information item with endpoint [owner element]** [p.57] .
- Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

- Zero or more *element information item* amongst its [children], in order, as follows:
  1. Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  2. Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl". Such *element information items* are considered to be endpoint extension elements as described below (see **2.13.2.4 Endpoint extension elements** [p.57]).

2.13.2.1 `name` *attribute information item* with `endpoint` [owner element]

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `description` *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*.

2.13.2.2 `binding` *attribute information item* with `endpoint` [owner element]

The `binding` *attribute information item* refers, by QName, to a Binding [p.39] 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*.

2.13.2.3 `address` *attribute information item* with `endpoint` [owner element]

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*.

2.13.2.4 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 [p.55] component with additional properties and specify the mapping from the XML representation to those properties.

2.13.3 Mapping Endpoint's XML Representation to Component Properties

The mapping from the XML Representation of the `endpoint` *element information item* (see **2.13.2 XML Representation of Endpoint Component** [p.55]) to the properties of the Endpoint [p.55] component is as described in Table 2-13 [p.57] .

Table 2-13. Mapping from XML Representation to Endpoint Component Properties

Property	Value
{name [p.55] }	The actual value of the `name` *attribute information item*.
{binding [p.55] }	The Binding [p.39] component resolved to by the actual value of the `binding` *attribute information item* (see **2.17 QName resolution** [p.59]).
{address [p.55] }	The actual value of the `address` *attribute information item* if present; otherwise empty.
{parent [p.55] }	The Service [p.52] component corresponding to the `service` *element information item* in [parent].

2.14 XML Schema 1.0 Simple Types Used in the Component Model

The XML Schema 1.0 simple types [*XML Schema: Datatypes [p.79]*] used in this specification are:

- *xs:token*
- *xs:NCName*

- *xs:anyURI*
- *xs:QName*
- *xs:boolean*

2.15 Equivalence of Components

Two component instances of the same type are considered equivalent if, for each property value of the first component, there is a corresponding property with an equivalent value on the second component, and vice versa.

- For values of a simple type (see **2.14 XML Schema 1.0 Simple Types Used in the Component Model** [p.57]) 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.78]*], or two boolean values are equivalent if they contain the same canonical value (`true` or `false`).
- 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, for each value in the first, there is an equivalent value in the second, and vice versa.

Extension properties which are not string values, sets of strings or references MUST describe their values' equivalence rules.† [p.100]

Because different top-level components (e.g., Interface [p.17] , Binding [p.39] , and Service [p.52]) are required to have different names, it is possible to determine whether two top-level components of a given type are equivalent by simply examining their {name} property.

The Binding [p.39] component specified by the {binding [p.55] } property of an Endpoint [p.55] is said to be applied to the Interface [p.17] component which is the value of the {interface [p.52] } property of the {parent [p.55] } Service [p.52] component for the Endpoint [p.55] . Note that, if this Binding [p.39] component has an {interface [p.39] } property, then its value MUST be the Interface [p.17] component that the Binding [p.39] component is applied to.

2.16 Symbol Spaces

This specification defines three symbol spaces, one for each top-level component type (Interface [p.17] , Binding [p.39] and Service [p.52]).

Within a symbol space, all qualified names (that is, the {name} property) are unique. Between symbol spaces, the names 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 2.0 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.78]*]. Other type systems may define additional symbol spaces.

2.17 QName resolution

In its serialized form WSDL 2.0 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. The namespace name for a component is represented by the value of the `targetNamespace` *attribute information item* of the [parent] `description` *element information item*. The local name is represented by the {name [p.58] } property of the component.

QName references are resolved by looking in the appropriate property of the Description [p.13] component. For example, to resolve a QName of an interface (as referred to by the `interface` *attribute information item* on a binding), the {interfaces [p.13] } property of the Description [p.13] component would be inspected.

If the appropriate property of the Description [p.13] component does not contain a component with the required QName, then the reference is a broken reference. A Description [p.13] component MUST NOT have such broken references. † [p.96]

2.18 Comparing URIs and IRIs

This specification uses absolute URIs and IRIs to identify several components and components characteristics (for example, operation message exchange patterns and styles). When such absolute URIs and IRIs are being compared to determine equivalence (see **2.15 Equivalence of Components** [p.58]), they MUST be compared character-by-character as indicated in [*IETF RFC 3987 [p.78]*]. † [p.99]

3. Types

```
<description>
  <types>
    <documentation />*
    [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
      <xs:schema targetNamespace="xs:anyURI"? /> |
      other extension elements ]*
  </types>
</description>
```

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 2.0 extensions), this specification only defines a means of expressing constraints based upon the XML Infoset [*XML Information Set [p.78]*]. 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 inlining or importing schema), this specification only defines the use of XML Schema [*XML Schema: Structures [p.78]*], [*XML Schema: Datatypes [p.79]*].

Specifically, the {element declarations [p.13] } and {type definitions [p.13] } properties of the Description [p.13] component are collections of imported and inlined 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 [*XML Schema: Structures [p.78]*], [*XML Schema: Datatypes [p.79]*] is included in the conformance criteria for WSDL 2.0 documents (see **3.1 Using W3C XML Schema Definition Language** [p.61]).

The schema components contained in the {element declarations [p.13] } property of the Description [p.13] component provide the type system used for Interface Message Reference [p.30] and Interface Fault [p.21] components. Interface Message Reference [p.30] 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 [p.21] components behave similarly. Such extensions should define how they reference type system components. Such type system components MAY appear in additional collection properties on the Description [p.13] component.

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 the Description [p.13] component.

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/ns/wsdl".
- Zero or more namespace qualified *attribute information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl
- Zero or more *element information item*s amongst its [children] as follows:
  - Zero or more `documentation` *element information item*s (see **5. Documentation** [p.71]) 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/ns/wsdl

3.1 Using W3C XML Schema Definition Language

XML Schema MAY be used as the schema language via import or inlining.

A WSDL 2.0 document MUST NOT refer to XML Schema components in a given namespace UNLESS an `xs:import` or `xs:schema` *element information item* for that namespace is present OR the namespace is the XML Schema namespace, http://www.w3.org/2001/XMLSchema, which contains built-in types as defined in XML Schema Part 2: Datatypes Second Edition [*XML Schema: Datatypes [p.79]*].[† [p.97]] That is, using the `xs:import` or `xs:schema` *element information item* is a necessary condition for making XML Schema components, other than the built-in components, referenceable within a WSDL 2.0 document. The built-in XML Schema datatypes are built-in to the WSDL 2.0 component model and are contained in the {type definitions [p.13] } property of the Description [p.13] component. A WSDL 2.0 document that refers to any element declaration or type definition component of the XML Schema namespace, except the built-in primitive and derived types, MUST import http://www.w3.org/2001/XMLSchema.

Table 3-1 [p.61] summarizes the referenceability of schema components.

Table 3-1. Referenceability of schema components

	XML Representation	**Referenceability of XML Schema Components**
Including description	description/include	XML Schema components in the included Description [p.13] component's {element declarations [p.13] } and {type definitions [p.13] } properties are referenceable.
Importing description	description/import	None of the XML Schema Components in the imported Description [p.13] component are referenceable.
Importing XML Schema	description/types/xs:import	Element Declaration [p.12] and Type Definition [p.12] components in the imported namespace are referenceable.
Inlined XML Schema	description/types/xs:schema	Element Declaration [p.12] and Type Definition [p.12] components in the inlined XML Schema are referenceable.

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.78]*], [*XML Schema: Datatypes [p.79]*], with the differences defined in this section and the following one. The schema components defined in the imported namespace are referenceable by QName (see **2.17 QName resolution** [p.59]). Only components in the imported namespace are referenceable in the WSDL 2.0 document. For each component in the imported namespace, a corresponding Element Declaration [p.12] component or Type Definition [p.12] component MUST appear in the {element declarations [p.13] } or {type definitions [p.13] } property respectively of the Description [p.13] component corresponding to the WSDL document that imports the schema, or that imports directly or indirectly a WSDL document that imports the schema.† [p.100] Schema components not in an imported namespace MUST NOT appear in the {element declarations [p.13] } or {type definitions [p.13] } properties.† [p.100]

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 items* 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*.† [p.97] The value of the `targetNamespace` *attribute information item* of the `xs:schema` *element information item* of an imported schema MUST equal the value of the `namespace` of the `import` *element information item* in the importing WSDL 2.0 document.† [p.97] Note that a WSDL 2.0 document must not 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 2.0 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 XML Schema 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*.

Every QName reference must resolve (see **2.17 QName resolution** [p.59]). Note that, when resolving QNames references for schema definitions, the namespace must be imported by the referring WSDL 2.0 document (see **3.1 Using W3C XML Schema Definition Language** [p.61]).

3.1.2 Inlining XML Schema

Inlining an XML schema uses the existing top-level `xs:schema` *element information item* defined by XML Schema [*XML Schema: Structures [p.78]*]. Conceptually, inlining can be viewed as simply cutting and pasting an existing schema document to a location inside the types *element information item*.

The schema components defined and declared in the inlined schema document are referenceable by QName (see **2.17 QName resolution** [p.59]). Only components defined and declared in the schema itself and components included by it via `xs:include` are referenceable. For each component defined and declared in the inlined schema document or included by `xs:include`, a corresponding Element Declaration [p.12] component or Type Definition [p.12] component MUST appear in the {element declarations [p.13] } property or {type definitions [p.13] } property respectively of the Description [p.13] component corresponding to the WSDL document that contains the schema, or that imports directly or indirectly a WSDL document that contains the schema. † [p.100] Schema components not defined or declared in the inlined schema document or included by `xs:include` MUST NOT appear in the {element declarations [p.13] } or {type definitions [p.13] } properties. † [p.100]

Note that components in the namespace that the inline schema imports via `xs:import` are not automatically referenceable from the WSDL 2.0 document that contains the inline schema. If the namespace referenced in a QName is contained in an inline schema, it MAY be imported without a `schemaLocation` attribute, so long as the inline schema has been resolved in the current component model.

Note that components defined in an inlined XML schema are not automatically referenceable within the WSDL 2.0 document that imported (using `wsdl:import`) the WSDL 2.0 document that inlines the schema (see **4.2 Importing Descriptions** [p.68] for more details). For this reason, it is recommended that XML schema documents intended to be shared across several WSDL 2.0 documents be placed in separate XML schema documents and imported using `xs:import`, rather than inlined inside a WSDL 2.0 document.

Inside an inlined XML schema, the `xs:import` and `xs:include` *element information item*s MAY be used to refer to other XML schemas inlined in the same or other WSDL 2.0 document, provided that an appropriate value, such as a fragment identifier (see [*XML Schema: Structures [p.78]*] 4.3.1) is specified for their `schemaLocation` *attribute information item*s. For `xs:import`, the `schemaLocation` attribute is not required so long as the namespace has been resolved in the current component model. The semantics of such *element information item*s are governed solely by the XML Schema specification [*XML Schema: Structures [p.78]*].

A WSDL 2.0 document MAY inline two or more schemas from the same `targetNamespace`. For example, two or more inlined schemas can have the same `targetNamespace` provided that they do not define the same elements or types. A WSDL 2.0 document MUST NOT define the same element or type in more than one inlined schema. † [p.97] Note that it is the responsibility of the underlying XML Schema processor to sort out a coherent set of schema components.

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".
- 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.

3.1.3 References to Element Declarations and Type Definitions

Whether inlined or imported, the global element declarations present in a schema are referenceable from an Interface Message Reference [p.30] or Interface Fault [p.21] component. Similarly, regardless of whether they are inlined or imported, the global type definitions present in a schema are referenceable from other components.

A named, global `xs:element` declaration is referenceable from the `element` *attribute information item* of an `input`, `output` (see **2.5.2 XML Representation of Interface Message Reference Component** [p.31]) or `fault` *element information item* (see **2.3.2 XML Representation of Interface Fault Component** [p.22]). The QName of the element declaration is constructed from the `targetNamespace` of the schema and the value of the `name` *attribute information item* of the `xs:element` *element information item*. Note that the `element` *attribute information item* cannot refer to a global `xs:simpleType` or `xs:complexType` definition, since these are in a different symbol space than global element declarations. If the `element` *attribute information item* erroneously contains the QName of a type definition then this would result in a failure to resolve the element declaration.

3.2 Using Other Schema Languages

Since it is unreasonable to expect that a single schema language can be used to describe all possible Interface Message Reference [p.30] and Interface Fault [p.21] component contents and their constraints, WSDL 2.0 allows alternate schema languages to be specified via extension elements. An extension *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 Interface Message Reference [p.30] and Interface Fault [p.21] components. Depending upon the schema language used, an *element information item* MAY be defined to allow inlining, 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*). † [p.97] The extension specification SHOULD, if necessary, define additional properties of the Description [p.13] component (and extension attributes) to hold the components of the referenced type system. It is expected that additional extension attributes for Interface Message Reference [p.30] and Interface Fault [p.21] components will also be defined, along with a mechanism for resolving the values of those attributes to a particular imported type system component.

A specification of extension syntax for an alternative schema language MUST use a namespace that is different than the namespace of XML Schema. † [p.97] The namespace of the alternative schema language is used for *element information item*s that are children of the `wsdl:types` *element information item* and for any extension *attribute information item*s that appear on other components. The namespace used for an alternate schema language MUST be an absolute IRI. † [p.97]

See [*WSDL 2.0 Alternative Schema Languages Support [p.80]*] for examples of using other schema languages. These examples reuse the {element declarations [p.13] } property of the Description [p.13] component and the `element` *attribute information item*s of the `wsdl:input`, `wsdl:output` and `wsdl:fault` *element information item*s.

Note:

This specification does not define the behavior of a WSDL 2.0 document that uses multiple schema languages for describing type system components simultaneously.

3.3 Describing Messages that Refer to Services and Endpoints

Web services can exchange messages that refer to other Web services or Web service endpoints. If the interface or binding of these referenced services or endpoints are known at description time, then it may be useful to include this information in the WSDL 2.0 document that describes the Web service. WSDL 2.0 provides two global *attribute information item*s, `wsdlx:interface` and `wsdlx:binding` that may be used to annotate XML Schema components or components from other type description languages.

WSDL 2.0 defines the use of these global *attribute information item*s to annotate XML Schema components that use the `xs:anyURI` simple type in an *element information item* or *attribute information item* for endpoint addresses that correspond to the {address [p.55] } property of the Endpoint [p.55] component. However, the use of these global *attribute information items* is not limited to simple types based on `xs:anyURI`. They may be used for any other types that are used to refer to Web services or Web service endpoints, e.g. a WS-Addressing Endpoint Reference [*WSA 1.0 Core [p.79]*]. See the primer [*WSDL 2.0 Primer [p.79]*] for more information and examples.

3.3.1 `wsdlx:interface` *attribute information item*

WSDL 2.0 provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `interface`.
- A [namespace name] of " http://www.w3.org/ns/wsdl-extensions ".

The type of the `wsdlx:interface` *attribute information item* is an *xs:QName* that specifies the {name [p.17] } property of an Interface [p.17] component.† [p.97]

3.3.2 `wsdlx:binding` *attribute information item*

WSDL 2.0 provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `binding`.
- A [namespace name] of " http://www.w3.org/ns/wsdl-extensions ".

The type of the `wsdlx:binding` *attribute information item* is an *xs:QName* that specifies the {name [p.39] } property of a Binding [p.39] component.† [p.97]

3.3.3 `wsdlx:interface` and `wsdlx:binding` Consistency

The `wsdlx:interface` and `wsdlx:binding` attributes may be used either independently or together. If `wsdlx:interface` and `wsdlx:binding` are used together then they MUST satisfy the same consistency rules that apply to the {interface [p.52] } property of a Service [p.52] component and the {binding [p.55] } property of a nested Endpoint [p.55] component, that is either the binding refers the interface of the service or the binding refers to no interface.† [p.97]

3.3.4 Use of `wsdlx:interface` and `wsdlx:binding` with `xs:anyURI`

`wsdlx:interface` and `wsdlx:binding` are used to describe *element information items* and *attribute information items* whose type is `xs:anyURI` or a restriction of it, as well messages that contain the {address [p.55] } property of an Endpoint [p.55] . This is accomplished by including the `wsdlx:interface` and/or `wsdlx:binding` *attribute information item* in the `xs:element`, `xs:simpleType`, or `xs:attribute` *element information item* of the corresponding XML Schema component.

4. Modularizing WSDL 2.0 descriptions

WSDL 2.0 provides two mechanisms for modularizing WSDL 2.0 descriptions. These mechanisms help to make Web service descriptions clearer by allowing separation of the various components of a description. Such separation can 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 even according to some other grouping such as application applicability.

Both mechanisms work at the level of WSDL 2.0 components and NOT at the level of XML Information Sets or XML 1.0 serializations.

4.1 Including Descriptions

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

The WSDL 2.0 `include` *element information item* allows separating the different components of a service definition, belonging to the same target namespace, into independent WSDL 2.0 documents.

The WSDL 2.0 `include` *element information item* is modeled after the XML Schema `include` *element information item* (see [*XML Schema: Structures [p.78]*], section 4.2.3 "References to schema components in the same namespace"). Specifically, it can be used to include components from WSDL 2.0 descriptions that share a target namespace with the including description. Components in the transitive closure of the included WSDL 2.0 documents become part of the Description [p.13] component of the including WSDL 2.0 document. The included components can be referenced by QName. Note that because all WSDL 2.0 descriptions have a target namespace, no-namespace includes (sometimes known as “chameleon includes”) never occur in WSDL 2.0.

A mutual include is the direct inclusion by one WSDL 2.0 document of another WSDL 2.0 document which includes the first document. A circular include achieves the same effect with greater indirection (for example, A includes B, B includes C, and C includes A). Multiple inclusion of a single WSDL 2.0 document resolves to a single set of components, as if the document was included only once. Mutual, multiple, and circular includes are explicitly permitted, and do not represent multiple redefinitions of the same components.

The `include` *element information item* has:

- A [local name] of `include`.
- A [namespace name] of "http://www.w3.org/ns/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 element]** [p.68] .
- Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

- Zero or more *element information item* amongst its [children], as follows:
  - Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

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

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 `description` *element information item*.

The IRI indicated by `location` MUST resolve to a WSDL 2.0 document.† [p.94] (see **7. Locating WSDL 2.0 Documents** [p.75])

The actual value of the `targetNamespace` *attribute information item* of the included WSDL 2.0 document MUST match the actual value of the `targetNamespace` *attribute information item* of the `description` *element information item* which is the [parent] of the `include` *element information item*.† [p.94]

4.2 Importing Descriptions

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

Every top-level WSDL 2.0 component is associated with a namespace. Every WSDL 2.0 document carries a `targetNamespace` *attribute information item* on its `wsdl:description` *element information item*. This attribute associates the document with a target namespace, which consequently also becomes the namespace of each top-level WSDL 2.0 component defined in that document. Any namespace other than the document's target namespace is referred to as a *foreign namespace*. Any component associated with a foreign namespace is referred to as a *foreign component*. This section describes the syntax and mechanisms by which references may be made from within a WSDL 2.0 document to foreign components. In addition to this syntax, there is an optional facility for suggesting the IRI of a WSDL 2.0 document that contains definitions of foreign components.

The WSDL 2.0 `import` *element information item* is modeled after the XML Schema `import` *element information item* (see [*XML Schema: Structures [p.78]*], section 4.2.3 "References to schema components across namespaces"). Specifically, it can be used to import WSDL 2.0 components from a foreign namespace. The WSDL 2.0 `import` *element information item* identifies a foreign namespace. The presence of a WSDL 2.0 `import` *element information item* signals that the WSDL 2.0 document may contain references to foreign components. The `wsdl:import` *element information item* is therefore like a forward declaration for foreign namespaces.

As with XML schema, any WSDL 2.0 document that references a foreign component MUST have a `wsdl:import` *element information item* for the associated foreign namespace (but which does not necessarily provide a `location` *attribute information item* that identifies the WSDL 2.0 document in which the referenced component is defined). † [p.94] In other respects, the visibility of components is pervasive: if two WSDL 2.0 documents import the same namespace, then they will have access to the same components from the imported namespace (i.e. regardless of which, if any, `location` *attribute information item* values are provided on the respective `wsdl:import` *element information items*.)

Using the `wsdl:import` *element information item* is a necessary condition for making foreign components available to a WSDL 2.0 document. That is, a WSDL 2.0 document can only refer to foreign components, if it contains an `wsdl:import` *element information item* for the associated foreign namespace.

If a WSDL 2.0 document contains more than one `wsdl:import` *element information item* for a given value of the `namespace` *attribute information item*, then they MUST provide different values for the `location` *attribute information item*. † [p.94] Repeating the `wsdl: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 dereferencable. When it is not dereferencable, no information about the imported namespace is provided by that `wsdl:import` *element information item*. It is possible that such lack of information can cause QNames in other parts of a WSDL 2.0 Description [p.13] component to become broken references (see **2.17 QName resolution** [p.59]). Such broken references are not ascribed to the `wsdl:import` *element information item*, but rather are failures of the QName resolution requirements which must be detected as described in **2.17 QName resolution** [p.59] .

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

- A [local name] of `import`.
- A [namespace name] of "http://www.w3.org/ns/wsdl".
- One or more *attribute information items* amongst its [attributes] as follows:
    - A REQUIRED `namespace` *attribute information item* as described below in **4.2.1 namespace attribute information item** [p.70] .

- An OPTIONAL `location` *attribute information item* as described below in **4.2.2 location attribute information item with import [owner element]** [p.70] .
- Zero or more namespace qualified *attribute information items* whose [namespace name] is NOT "http://www.w3.org/ns/wsdl".

- Zero or more *element information items* amongst its [children], as follows:
  - Zero or more `documentation` *element information items* (see **5. Documentation** [p.71]).
  - Zero or more namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/ns/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 2.0 document MAY contain qualified references to WSDL 2.0 components 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 `targetNamespace` *attribute information item* in the enclosing WSDL 2.0 document.† [p.94] If the location attribute in the `import` *element information item* is dereferencable, then it MUST reference a WSDL 2.0 document.† [p.94] If the `location` *attribute information item* of the `import` *element information item* is dereferencable, then the actual value of the `namespace` *attribute information item* MUST be identical to the actual value of the `targetNamespace` *attribute information item* of the referenced WSDL 2.0 document (see **7. Locating WSDL 2.0 Documents** [p.75]).† [p.94]

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

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, if present, gives a hint as to where a serialization of a WSDL 2.0 document with definitions for the imported namespace can be found.

The `location` *attribute information item* is optional. This allows WSDL 2.0 components to be constructed from information other than an XML 1.0 serialization of a WSDL 2.0 document. It also allows the development of WSDL 2.0 processors that have *a prior* (i.e., built-in) knowledge of certain namespaces.

4.3 Extensions

The semantics of an extension MUST NOT depend on how components are brought into a component model instance via <import> or <include>. † [p.101] That is, the components that are defined by a WSDL 2.0 document are determined by the contents of the document, EXCEPT for the resolution of references to other components that may be defined in other documents, AND any further processing, as mandated by the extension specification, that depends on such references having been resolved to the actual components.

This restriction on the behavior of extensions permits WSDL 2.0 documents to be flexibly modularized and efficiently processed. In contrast, note that the so-called chameleon include mechanism of XML Schema, which allows a no-namespace schema to be included in a schema document that has a namespace, violates this restriction since the namespace of the included XML Schema components is determined by the including XML Schema document (see 4.2.1 Assembling a schema for a single target namespace from multiple schema definition documents in [*XML Schema: Structures [p.78]*]).

5. Documentation

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

WSDL 2.0 uses the optional `documentation` *element information item* as a container for human readable or machine processable documentation. The content of the *element information item* is arbitrary *character information items* and *element information items* ("mixed" content in XML Schema [*XML Schema: Structures [p.78]*]). The `documentation` *element information item* is allowed inside any WSDL 2.0 *element information item*.

Like other *element information items* in the "http://www.w3.org/ns/wsdl" namespace, the `documentation` *element information item* allows qualified *attribute information items* whose [namespace name] is not "http://www.w3.org/ns/wsdl". The `xml:lang` attribute (see [*XML 1.0 [p.78]*]) MAY be used to indicate the language used in the contents of the `documentation` *element information item*.

The `documentation` *element information item* has:

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

6. Language Extensibility

The schema for WSDL 2.0 has a two-part extensibility model based on namespace-qualified elements and attributes. An extension is identified by the QName consisting of its namespace IRI and its element or attribute name. The meaning of an extension SHOULD be defined (directly or indirectly) in a document that is available at its namespace IRI. † [p.101]

6.1 Element-based Extensibility

WSDL 2.0 allows extensions to be defined in terms of *element information item*s. Where indicated herein, WSDL 2.0 allows namespace-qualified *element information item*s whose [namespace name] is NOT "http://www.w3.org/ns/wsdl" to appear among the [children] of specific *element information item*s whose [namespace name] is "http://www.w3.org/ns/wsdl". Such *element information item*s MAY be used to annotate WSDL 2.0 constructs such as interface, operation, etc.

It is expected that extensions will 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 from the XML representation of the extension to the properties in the component model.

The WSDL 2.0 schema defines a base type for use by extension elements. Example 6-1 [p.72] shows the type definition. The use of this type as a base type is optional.

Example 6-1. Base type for extension elements

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

Extension 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 2.0 specification. WSDL 2.0 recommends that specifications defining such protocols also define any necessary WSDL 2.0 extensions used to describe those protocols or formats.

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.74]) 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 a WSDL 2.0 document. † [p.100] 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 2.0 document.

If a WSDL 2.0 document declares an extension as optional (i.e., NON-mandatory), then the Web service MUST NOT assume that the client supports that extension *unless* the Web service knows (through some other means) that the client has in fact elected to engage and support that extension. † [p.101]

Note:

A key purpose of an extension is to formally indicate (i.e., in a machine-processable way) that a particular feature or convention is supported or required. This enables toolkits that understand the extension to engage it automatically, while toolkits that do not yet understand a required extension can possibly bring it to the attention of an operator for manual support.

If a Web service requires a client to follow a particular convention that is likely to be automatable in WSDL 2.0 toolkits, then that convention SHOULD be indicated in the WSDL 2.0 document as a `wsdl:required` extension, rather than just being conveyed out of band, even if that convention is not currently implemented in WSDL 2.0 toolkits.

This practice will help prevent interoperability problems that could arise if one toolkit requires a particular convention that is not indicated in the WSDL 2.0 document, while another toolkit does not realize that that convention is required. It will also help facilitate future automatic processing by WSDL 2.0 toolkits.

On the other hand, a client MAY engage an extension that is declared as optional in the WSDL 2.0 document. Therefore, the Web service MUST support every extension that is declared as optional in the WSDL 2.0 document, in addition to supporting every extension that is declared as mandatory. † [p.101]

Note:

If finer-grain, direction-sensitive control of extensions is desired, then such extensions may be designed in a direction-sensitive manner (from the client or from the Web service) 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.

Validity of a WSDL 2.0 document can only be assessed within the context of a set of supported extensions. A WSDL 2.0 document that contains a required but unsupported extension is invalid with respect to that set of supported extensions.

6.1.2 `required` *attribute information item*

WSDL 2.0 provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `required`.
- A [namespace name] of "http://www.w3.org/ns/wsdl".

The type of the `required` *attribute information item* is *xs:boolean*. Its default value is "false" (hence extensions are NOT required by default).

6.2 Attribute-based Extensibility

WSDL 2.0 allows qualified *attribute information item*s whose [namespace name] is NOT "http://www.w3.org/ns/wsdl" to appear on any *element information item* whose namespace name IS "http://www.w3.org/ns/wsdl". Such *attribute information item*s can be used to annotate WSDL 2.0 constructs such as interfaces, bindings, etc.

WSDL 2.0 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 extension elements and attributes will result in additional properties appearing in the component model.

The presence of an optional extension element or attribute MAY therefore augment the semantics of a WSDL 2.0 document in ways that do not invalidate the existing semantics. However, the presence of a mandatory extension element MAY alter the semantics of a WSDL 2.0 document in ways that invalidate the existing semantics.

Extension elements SHOULD NOT alter the existing semantics in ways that are likely to confuse users.

Note:

Note that, however, once the client and service both know that an optional extension has been engaged (because the service has received a message explicitly engaging that extension, for example), then the semantics of that extension supersede what the WSDL 2.0 document indicated. For example, the WSDL 2.0 document may have specified an XML message schema to be used, but also indicated an optional security extension that encrypts the messages. If the security extension is engaged, then the encrypted messages will no longer conform to the specified message schema (until they are decrypted).

Note:

Authors of extension elements should make sure to include in the specification of these elements a clear statement of the requirements for document conformance (see **1.3 Document Conformance** [p.7]).

Note:

Authors of extension elements that may manifest as properties of the Description [p.13] component should be aware of the impact of imports on their extensions, or of their extensions on imports. It is not possible, within the component model, to define extensions that have an effective scope equal to the scope of a containing file. Extensions that modify the behavior of the components contained in a description may therefore unexpectedly modify the behavior of components in imported descriptions as well, unless proper care is taken.

7. Locating WSDL 2.0 Documents

A *WSDL 2.0 document* is a `description` *element information item* that is either the document root of an XML document or an element within an XML document. The *location* of a WSDL 2.0 MAY therefore be specified by an IRI for an XML resource whose document root is a `description` *element information item* or an IRI-reference for a `description` *element information item* within an XML resource.

As an XML vocabulary, WSDL 2.0 documents, WSDL2.0 document fragments or QName references to WSDL 2.0 components MAY appear within other XML documents. This specification defines a global attribute, `wsdlLocation`, to help with QName resolution (see **2.17 QName resolution** [p.59]). This attribute allows an element that contains such references to be annotated to indicate where the WSDL 2.0 documents for one or more namespaces can be found. In particular, this attribute is expected to be useful when using service references in message exchanges.

The `wsdlLocation` global attribute is defined in the namespace "http://www.w3.org/ns/wsdl-instance" (hereafter referred to as "wsdli:wsdlLocation", for brevity). This attribute MAY appear on any XML element which allows attributes from other namespaces to occur. It MUST NOT appear on a `wsdl:description` element or any of its children/descendants. † [p.95]

A normative XML Schema [*XML Schema: Structures [p.78]*], [*XML Schema: Datatypes [p.79]*] document for the "http://www.w3.org/ns/wsdl-instance" namespace can be found at http://www.w3.org/ns/wsdl-instance.

7.1 `wsdli:wsdlLocation` *attribute information item*

WSDL 2.0 provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `wsdlLocation`.
- A [namespace name] of "http://www.w3.org/ns/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 IRIs; where the first IRI of a pair, which MUST be an absolute IRI as defined in [*IETF RFC 3987 [p.78]*], indicates a WSDL 2.0 (or 1.1) namespace name, and, the second a hint as to the location of a WSDL 2.0 document defining WSDL 2.0 components (or WSDL 1.1 elements [*WSDL 1.1 [p.79]*]) for that namespace name. † [p.103] The second IRI of a pair MAY be absolute or relative. For each pair of IRIs, if the location IRI of the pair is dereferencable, then it MUST reference a WSDL 2.0 (or 1.1) document whose target namespace is the namespace IRI of the pair. † [p.95]

8. Conformance

This section describes how this specification conforms to other specifications. This is limited, at present, to the XML Information Set specification. Refer to **1.3 Document Conformance** [p.7] for a description of the criteria that Web service description documents must satisfy in order to conform to this specification.

8.1 XML Information Set Conformance

This specification conforms to the [*XML Information Set [p.78]*]. The following information items MUST be present in the input Infosets to enable correct processing of WSDL 2.0 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.

9. XML Syntax Summary (Non-Normative)

```
<description targetNamespace="xs:anyURI" >
  <documentation />*

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

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

  <types>
    <documentation />*

      [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
        <xs:schema targetNamespace="xs:anyURI"? /> |
        other extension elements ]*
  </types>

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

    <fault name="xs:NCName" element="union of xs:QName, xs:token"? >
      <documentation />*
    </fault>*

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

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

```
      </input>*

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

      </output>*

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

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

    </operation>*

  </interface>*

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

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

    <operation ref="xs:QName" >
      <documentation />*

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

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

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

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

    </operation>*

  </binding>*

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

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

  </service>*
</description>
```

10. References

10.1 Normative References

[Character Model for the WWW]
Character Model for the World Wide Web 1.0: Fundamentals, M. Dürst, F. Yergeau, R. Ishida, M. Wolf, T. Texin, Editors. W3C Recommendation, 15 February 2005. Latest version available at http://www.w3.org/TR/charmod/.

[IETF RFC 2119]
Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt.

[IETF RFC 3023]
XML Media Types, M. Murata, S. St. Laurent, D. Kohn, Authors. Internet Engineering Task Force, January 2001. Available at http://www.ietf.org/rfc/rfc3023.txt.

[IETF RFC 3986]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3986.txt.

[IETF RFC 3987]
Internationalized Resource Identifiers (IRIs), M. Duerst, M. Suignard, Authors. Internet Engineering Task Force, January 2005. Available at http://www.ietf.org/rfc/rfc3987.txt.

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

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Fourth Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, Editors. World Wide Web Consortium, 10 February 1998, revised 16 August 2006. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2006/REC-xml-20060816/. The latest version of "Extensible Markup Language (XML) 1.0" is available at http://www.w3.org/TR/REC-xml.

[XML Namespaces]
Namespaces in XML (Second Edition), T. Bray, D. Hollander, A. Layman, and R. Tobin, Editors. World Wide Web Consortium, 16 August 2006. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2006/REC-xml-names-20060816. The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names.

[XML Information Set]
XML Information Set (Second Edition), J. Cowan and R. Tobin, Editors. World Wide Web Consortium, 24 October 2001, revised 4 February 2004. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2004/REC-xml-infoset-20040204. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset.

[XML Schema: Structures]
XML Schema Part 1: Structures Second Edition, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. This version of the XML Schema Part 1 Recommendation is

http://www.w3.org/TR/2004/REC-xmlschema-1-20041028. 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 Second Edition, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 2 May 2001, revised 28 October 2004. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2004/REC-xmlschema-2-20041028. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2.

[WSDL 2.0 Adjuncts]
Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts , R. Chinnici, H. Haas, A. Lewis, J-J. Moreau, D. Orchard, S. Weerawarana, Editors. World Wide Web Consortium, 26 June 2007. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts" Recommendation is available at http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts" is available at http://www.w3.org/TR/wsdl20-adjuncts.

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.

[WSA 1.0 Core]
Web Services Addressing 1.0 - Core , M. Gudgin, M. Hadley, T. Rogers, Editors. World Wide Web Consortium, 9 May 2006. This version of Web Services Addressing 1.0 - Core Recommendation is http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ The latest version of the "Web Services Addressing 1.0 - Core" document is available from http://www.w3.org/TR/ws-addr-core.

[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 Language (WSDL) Version 2.0 Part 0: Primer , D.Booth, C.K. Liu , Editors. World Wide Web Consortium, 26 June 2007. This version of the "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" Recommendation is available at http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" is available at http://www.w3.org/TR/wsdl20-primer.

[WSDL 2.0 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.

[WSDL 2.0 Alternative Schema Languages Support]
Discussion of Alternative Schema Languages and Type System Support in WSDL 2.0, A. Lewis, B. Parsia, Editors. World Wide Web Consortium, 17 August 2005. This version of the "Discussion of Alternative Schema Languages and Type System Support in WSDL 2.0" Working Group Note is http://www.w3.org/TR/2005/NOTE-wsdl20-altschemalangs-20050817/. The latest version of "Discussion of Alternative Schema Languages and Type System Support in WSDL 2.0" is available at http://www.w3.org/TR/wsdl20-altschemalangs.

[SOAP 1.2 Part 1: Messaging Framework (Second Edition)]
SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), M. Gudgin, et al., Editors. World Wide Web Consortium, 24 June 2003, revised 27 April 2007. This version of the "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)" Recommendation is http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. The latest version of "SOAP Version 1.2 Part 1: Messaging Framework" is available at http://www.w3.org/TR/soap12-part1/.

[XPointer]
XPointer Framework, P. Grosso, E. Maler, J. Marsh, N. Walsh, Editors. World Wide Web Consortium, 25 March 2003. This version of the XPointer Framework 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/.

[Z Notation Reference Manual]
The Z Notation: A Reference Manual, Second Edition, J. M. Spivey, Prentice Hall, 1992.

[Fuzz 2000]
Release Notes For Fuzz 2000, J. M. Spivey.

A. The application/wsdl+xml Media Type

This appendix defines the "application/wsdl+xml" media type which can be used to describe WSDL 2.0 documents serialized as XML.

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 [*IETF RFC 3023 [p.78]*].

Encoding considerations:

Identical to those of "application/xml" as described in [*IETF RFC 3023 [p.78]*], section 3.2, as applied to the WSDL document Infoset.

Security considerations:

See section **A.3 Security considerations** [p.87] .

Interoperability considerations:

There are no known interoperability issues.

Published specifications:

This document and [*WSDL 2.0 Adjuncts [p.79]*].

Applications which use this media type:

No known applications currently use this media type.

Additional information:
File extension:

wsdl

Fragment identifiers:

Either a syntax identical to that of "application/xml" as described in [*IETF RFC 3023 [p.78]*], section 5 or the syntax defined in **A.2 Fragment Identifiers** [p.82] .

Base URI:

As specified in [*IETF RFC 3023 [p.78]*], section 6.

Macintosh File Type code:

WSDL

Person and email address to contact for further information:

World Wide Web Consortium <web-human@w3.org>

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 Fragment Identifiers

This section defines a fragment identifier syntax for identifying components of a WSDL 2.0 document. This fragment identifier syntax is compliant with the [*XPointer [p.80]*].

A WSDL 2.0 fragment identifier is an XPointer [*XPointer [p.80]*], augmented with WSDL 2.0 pointer parts as defined below. Note that many of these parts require the pre-appearance of one or more `xmlns` pointer parts (see *3.4 Namespace Binding Context* in [*XPointer [p.80]*]). The pointer parts have a scheme name that corresponds to one of the standard WSDL 2.0 component types, and scheme data that is a path composed of names that identify the components. The scheme names all begin with the prefix "wsdl." to avoid name conflicts with other schemes. The names in the path are of type either QName, NCName, IRI, URI, or Pointer Part depending on the context. The scheme data for WSDL 2.0 extension components is defined by the corresponding extension specification.

For QNames, any prefix MUST be defined by a preceding xmlns pointer part.† [p.101] If a QName does not have a prefix then its namespace name is the target namespace of the WSDL 2.0 document.

The fragment identifier is typically constructed from the {name [p.58] } property of the component and the {name [p.58] } properties of its ancestors as a path according to Table A-1 [p.82] . The first column of this table gives the name of the WSDL 2.0 component. Columns labeled 1 through 4 specify the identifiers that uniquely identify the component within its context. Identifiers are typically formed from the {name [p.58] } property, although in several cases references to other components are used. These identifiers are then used to construct the pointer part in the last column. The fragment identifier in a WSDL 2.0 component IRI-reference MUST resolve to some component as defined by the construction rules in Table A-1 [p.82] .† [p.101]

Table A-1. Rules for determining pointer parts for WSDL 2.0 components

Component	1	2	3	4	Pointer Part
Description [p.13]	n/a	n/a	n/a	n/a	`wsdl.description [p.84] ()`
Element Declaration [p.12]	*`element`* QName	n/a	n/a	n/a	`wsdl.elementDeclaration [p.84] (`*`element`*`)`
Element Declaration [p.12]	*`element`* QName	*`system`* IRI	n/a	n/a	`wsdl.elementDeclaration [p.84] (`*`element`*`,`*`system`*`)`

Type Definition [p.12]	*type* QName	n/a	n/a	n/a	wsdl.typeDefinition [p.84] (*type*)
Type Definition [p.12]	*type* QName	*system* IRI	n/a	n/a	wsdl.typeDefinition [p.84] (*type*,*system*)
Interface [p.17]	*interface* NCName	n/a	n/a	n/a	wsdl.interface [p.84] (*interface*)
Interface Fault [p.21]	*interface* NCName	*fault* NCName	n/a	n/a	wsdl.interfaceFault [p.84] (*interface*/*fault*)
Interface Operation [p.25]	*interface* NCName	*operation* NCName	n/a	n/a	wsdl.interfaceOperation [p.84] (*interface*/*operation*)
Interface Message Reference [p.30]	*interface* NCName	*operation* NCName	*message* NCName	n/a	wsdl.interfaceMessageReference [p.85] (*interface*/*operation*/*message*)
Interface Fault Reference [p.35]	*interface* NCName	*operation* NCName	*message* NCName	*fault* QName	wsdl.interfaceFaultReference [p.85] (*interface*/*operation*/*message*/*fault*)
Binding [p.39]	*binding* NCName	n/a	n/a	n/a	wsdl.binding [p.85] (*binding*)
Binding Fault [p.42]	*binding* NCName	*fault* QName	n/a	n/a	wsdl.bindingFault [p.85] (*binding*/*fault*)
Binding Operation [p.44]	*binding* NCName	*operation* QName	n/a	n/a	wsdl.bindingOperation [p.86] (*binding*/*operation*)
Binding Message Reference [p.47]	*binding* NCName	*operation* QName	*message* NCName	n/a	wsdl.bindingMessageReference [p.86] (*binding*/*operation*/*message*)
Binding Fault Reference [p.49]	*binding* NCName	*operation* QName	*message* NCName	*fault* QName	wsdl.bindingFaultReference [p.86] (*binding*/*operation*/*message*/*fault*)
Service [p.52]	*service* NCName	n/a	n/a	n/a	wsdl.service [p.86] (*service*)
Endpoint [p.55]	*service* NCName	*endpoint* NCName	n/a	n/a	wsdl.endpoint [p.87] (*service*/*endpoint*)
Extensions [p.12]	*namespace* URI	*identifier* extension-specific-syntax	n/a	n/a	wsdl.extension [p.87] (*namespace*,*identifier*)

Note that the above rules are defined in terms of component properties rather than the XML Infoset representation of the component model. The following sections specify in detail how the pointer parts are constructed from the component model.

A.2.1 The Description Component

`wsdl.description()`

A.2.2 The Element Declaration Component

`wsdl.elementDeclaration(`*`element`*`)`

`wsdl.elementDeclaration(`*`element`*`,`*`system`*`)`

1. *`element`* is the {name [p.12] } property of the Element Declaration [p.12] component.

2. *`system`* is the namespace absolute IRI of the extension type system used for the Element Declaration [p.12] component (see **3.2 Using Other Schema Languages** [p.65]). This parameter is absent if XML Schema is the type system.

A.2.3 The Type Definition Component

`wsdl.typeDefinition(`*`type`*`)`

`wsdl.typeDefinition(`*`type`*`,`*`system`*`)`

1. *`type`* is the {name [p.12] } property of the Type Definition [p.12] component.

2. *`system`* is the namespace absolute IRI of the extension type system used for the Type Definition [p.12] component (see **3.2 Using Other Schema Languages** [p.65]). This parameter is absent if XML Schema is the type system.

A.2.4 The Interface Component

`wsdl.interface(`*`interface`*`)`

1. *`interface`* is the local name of the {name [p.17] } property of the Interface [p.17] component.

A.2.5 The Interface Fault Component

`wsdl.interfaceFault(`*`interface`*`/`*`fault`*`)`

1. *`interface`* is the local name of the {name [p.17] } property of the parent Interface [p.17] component.

2. *`fault`* is the local name of the {name [p.21] } property of the Interface Fault [p.21] component.

A.2.6 The Interface Operation Component

`wsdl.interfaceOperation(`*`interface`*`/`*`operation`*`)`

1. *`interface`* is the local name of the {name [p.17] } property of the parent Interface [p.17] component.

2. *`operation`* is the local name of the {name [p.25] } property of the Interface Operation [p.25] component.

A.2.7 The Interface Message Reference Component

`wsdl.interfaceMessageReference(`*`interface/operation/message`*`)`

1. *`interface`* is the local name of the {name [p.17] } property of the grandparent Interface [p.17] component.

2. *`operation`* is the local name of the {name [p.25] } property of the parent Interface Operation [p.25] component.

3. *`message`* is the {message label [p.30] } property of the Interface Message Reference [p.30] component.

A.2.8 The Interface Fault Reference Component

`wsdl.interfaceFaultReference(`*`interface/operation/message/fault`*`)`

1. *`interface`* is the local name of the {name [p.17] } property of the grandparent Interface [p.17] component.

2. *`operation`* is the local name of the {name [p.25] } property of the parent Interface Operation [p.25] component.

3. *`message`* is the {message label [p.35] } property of the Interface Fault Reference [p.35] component.

4. *`fault`* is the {name [p.21] } property of the Interface Fault [p.21] component referred to by the {interface fault [p.35] } property of the Interface Fault Reference [p.35] component.

A.2.9 The Binding Component

`wsdl.binding(`*`binding`*`)`

1. *`binding`* is the local name of the {name [p.39] } property of the Binding [p.39] component.

A.2.10 The Binding Fault Component

`wsdl.bindingFault(`*`binding/fault`*`)`

1. *`binding`* is the local name of the {name [p.39] } property of the parent Binding [p.39] component.

2. *`fault`* is the {name [p.21] } property of the Interface Fault [p.21] component referred to by the {interface fault [p.42] } property of the Binding Fault [p.42] component.

A.2.11 The Binding Operation Component

`wsdl.bindingOperation(`*`binding/operation`*`)`

1. *`binding`* is the local name of the {name [p.39] } property of the parent Binding [p.39] component.
2. *`operation`* is the {name [p.25] } property of the Interface Operation [p.25] component referred to by the {interface operation [p.44] } property of the Binding Operation [p.44] component.

A.2.12 The Binding Message Reference Component

`wsdl.bindingMessageReference(`*`binding/operation/message`*`)`

1. *`binding`* is the local name of the {name [p.39] } property of the grandparent Binding [p.39] component.
2. *`operation`* is the {name [p.25] } property of the Interface Operation [p.25] component referred to by the {interface operation [p.44] } property of the parent Binding Operation [p.44] component.
3. *`message`* is the {message label [p.30] } property of the Interface Message Reference [p.30] component referred to by the {interface message reference [p.47] } property of the Binding Message Reference [p.47] component.

A.2.13 The Binding Fault Reference Component

`wsdl.bindingFaultReference(`*`binding/operation/message/fault`*`)`

1. *`binding`* is the local name of the {name [p.39] } property of the grandparent Binding [p.39] component.
2. *`operation`* is the {name [p.25] } property of the Interface Operation [p.25] component referred to by the {interface operation [p.44] } property of the parent Binding Operation [p.44] component.
3. *`message`* is the {message label [p.35] } property of the Interface Fault Reference [p.35] component referred to by the {interface fault reference [p.49] } property of the Binding Fault Reference [p.49] component.
4. *`fault`* is the {name [p.21] } property of the Interface Fault [p.21] component referred to by the {interface fault [p.35] } property of the Interface Fault Reference [p.35] component referred to by the {interface fault reference [p.49] } property of the Binding Fault Reference [p.49] component.

A.2.14 The Service Component

`wsdl.service(`*`service`*`)`

1. *`service`* is the local name of the {name [p.52] } property of the Service [p.52] component.

A.2.15 The Endpoint Component

`wsdl.endpoint(`*`service/endpoint`*`)`

1. *`service`* is the local name of the {name [p.52] } property of the parent Service [p.52] component.

2. *`endpoint`* is the {name [p.55] } property of the Endpoint [p.55] component.

A.2.16 Extension Components

WSDL 2.0 is extensible and it is possible for an extension to define new components types. The XPointer Framework scheme for extension components is:

`wsdl.extension(`*`namespace, identifier`*`)`

1. *`namespace`* is the namespace URI that identifies the extension, e.g. for the WSDL 2.0 SOAP 1.2 Binding the namespace is http://www.w3.org/ns/wsdl/soap.

2. *`identifier`* is defined by the extension using a syntax specific to the extension. The owner of the extension must define any components contributed by the extension and a syntax for identifying them.

A.3 Security considerations

This media type uses the "+xml" convention, it shares the same security considerations as described in [*IETF RFC 3023 [p.78]*], 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): Charlton Barreto (Adobe Systems, Inc), Allen Brookes (Rogue Wave Softwave), Dave Chappell (Sonic Software), Helen Chen (Agfa-Gevaert N. V.), Roberto Chinnici (Sun Microsystems), Kendall Clark (University of Maryland), Glen Daniels (Sonic Software), Paul Downey (British Telecommunications), Youenn Fablet (Canon), Ram Jeyaraman (Microsoft), Tom Jordahl (Adobe Systems), Anish Karmarkar (Oracle Corporation), Jacek Kopecky (DERI Innsbruck at the Leopold-Franzens-Universität Innsbruck, Austria), Amelia Lewis (TIBCO Software, Inc.), Philippe Le Hegaret (W3C), Michael Liddy (Education.au Ltd.), Kevin Canyang Liu (SAP AG), Jonathan Marsh (WSO2), Monica Martin (Sun Microsystems), Josephine Micallef (SAIC - Telcordia Technologies), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Cyclone Commerce), Jean-Jacques Moreau (Canon), David Orchard (BEA Systems, Inc.), Gilbert Pilz (BEA Systems, Inc.), Tony Rogers (Computer Associates), Arthur Ryman (IBM), Adi Sakala (IONA Technologies), Michael Shepherd (Xerox), Asir Vedamuthu (Microsoft Corporation), Sanjiva Weerawarana (WSO2), Ümit Yalçınalp (SAP AG), Peter Zehler (Xerox).

Previous members were: Eran Chinthaka (WSO2), Mark Nottingham (BEA Systems, Inc.), Hugo Haas (W3C), Vivek Pandey (Sun Microsystems), Bijan Parsia (University of Maryland), 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 Hégaret (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.), Ümit Yalçınalp (Oracle Corporation), Peter Madziak (Agfa-Gevaert N. V.), Jeffrey Schlimmer (Microsoft Corporation), Hao He (The Thomson Corporation), Erik Ackerman (Lexmark), Jerry Thrasher (Lexmark), Prasad Yendluri (webMethods, Inc.), William Vambenepe (Hewlett-Packard Company), David Booth (W3C), Sanjiva Weerawarana (IBM), Asir Vedamuthu (webMethods, Inc.), Igor Sedukhin (Computer Associates), Martin Gudgin (Microsoft Corporation), Rebecca Bergersen (IONA Technologies), Ugo Corda (SeeBeyond).

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

C. IRI-References for WSDL 2.0 Components (Non-Normative)

This appendix provides a syntax for IRI-references for all components found in a WSDL 2.0 document. The IRI-references are easy to understand and compare, while imposing no burden on the WSDL 2.0 author.

C.1 WSDL 2.0 IRIs

There are two main cases for WSDL 2.0 IRIs:

- the IRI of a WSDL 2.0 document
- the IRI of a WSDL 2.0 namespace

The IRI of a WSDL 2.0 document can be dereferenced to give a resource representation that contributes component definitions to a single WSDL 2.0 namespace. If the media type is set to the WSDL 2.0 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 Description Component** [p.12] that the namespace URI be dereferencable to a WSDL 2.0 document, this appendix specifies the use of the namespace IRI with the WSDL 2.0 fragment identifiers to form an IRI-reference.

The IRI in an IRI-reference for a WSDL 2.0 component is the namespace name of the {name [p.58] } property of either the component itself, in the case of Interface [p.17] , Binding [p.39] , and Service [p.52] components, or the {name [p.58] } property of the ancestor top-level component. The IRI provided by the namespace name of the {name [p.58] } property is combined with a zero or more `xmlns` pointer parts (see *3.4 Namespace Binding Context* in [*XPointer [p.80]*]) followed by a single WSDL 2.0 pointer part as defined in **A.2 Fragment Identifiers** [p.82] .

C.2 Canonical Form for WSDL 2.0 Component Designators

The IRI-references described above MAY be used as WSDL 2.0 component designators. For ease of comparison, the fragment identifier of WSDL 2.0 component designators SHOULD conform to the following canonicalization rules:

- The fragment identifier consists of a sequence zero or more `xmlns()` pointer parts followed by exactly one `wsdl.*()` pointer part. † [p.99]
- Each `xmlns()` pointer part that appears in the fragment identifier defines a namespace that is referenced by the `wsdl.*()` pointer part. † [p.99]
- Each `xmlns()` pointer part defines a unique namespace. † [p.99]
- The `xmlns()` pointer parts define namespaces in the same order as they are referenced in the `wsdl.*()` pointer part. † [p.99]
- The namespace prefixes defined by the `xmlns()` pointer parts are named `ns1` , `ns2` , etc., in the order of their appearance. † [p.99]
- The fragment identifier contains no optional whitespace. † [p.99]
- No `xmlns()` pointer part defines a namespace for the targetNamespace of the WSDL 2.0 document. † [p.99]

C.3 Example

Consider the following WSDL 2.0 document located at http://example.org/TicketAgent.wsdl:

Example C-1. IRI-References - Example WSDL 2.0 Document

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/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/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
        </wsdl:operation>

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

Its components have the following IRI-references which follow the above canonicalization rules except for the presence of optional whitespace that has been added in order to improve the formatting:

Example C-2. IRI-References - Example IRIs

```
http://example.org/TicketAgent.wsdl20#
  wsdl.description()

http://example.org/TicketAgent.wsdl20#
  xmlns(ns1=http://example.org/TicketAgent.xsd)
  wsdl.elementDeclaration(ns1:listFlightsRequest)

http://example.org/TicketAgent.wsdl20#
  xmlns(ns1=http://example.org/TicketAgent.xsd)
  wsdl.elementDeclaration(ns1:listFlightsResponse)

http://example.org/TicketAgent.wsdl20#
  xmlns(ns1=http://example.org/TicketAgent.xsd)
  wsdl.elementDeclaration(ns1:reserveFlightRequest)

http://example.org/TicketAgent.wsdl20#
  xmlns(ns1=http://example.org/TicketAgent.xsd)
  wsdl.elementDeclaration(ns1:reserveFlightResponse)

http://example.org/TicketAgent.wsdl20#
  wsdl.interface(TicketAgent)

http://example.org/TicketAgent.wsdl20#
  wsdl.interfaceOperation(TicketAgent/listFlights)

http://example.org/TicketAgent.wsdl20#
  wsdl.interfaceMessageReference(TicketAgent/listFlights/In)

http://example.org/TicketAgent.wsdl20#
  wsdl.interfaceMessageReference(TicketAgent/listFlights/Out)

http://example.org/TicketAgent.wsdl20#
  wsdl.interfaceOperation(TicketAgent/reserveFlight)

http://example.org/TicketAgent.wsdl20#
```

```
  wsdl.interfaceMessageReference(TicketAgent/reserveFlight/In)

http://example.org/TicketAgent.wsdl20#
  wsdl.interfaceMessageReference(TicketAgent/reserveFlight/Out)
```

D. Component Summary (Non-Normative)

Table D-1 [p.91] lists all the components in the WSDL 2.0 abstract Component Model, and all their properties. Note some properties have a generic definition that is used in more than one component. In this case, the Component column contains a "-" to indicate this generic definition of the property.

Table D-1. Summary of WSDL 2.0 Components and their Properties

Component	Defined Properties
-	{name [p.58] }, {parent [p.13] }
Binding [p.39]	{binding faults [p.39] }, {binding operations [p.39] }, {interface [p.39] }, {name [p.39] }, {type [p.39] }
Binding Fault [p.42]	{interface fault [p.42] }, {parent [p.42] }
Binding Fault Reference [p.49]	{interface fault reference [p.49] }, {parent [p.49] }
Binding Message Reference [p.47]	{interface message reference [p.47] }, {parent [p.47] }
Binding Operation [p.44]	{binding fault references [p.44] }, {binding message references [p.44] }, {interface operation [p.44] }, {parent [p.44] }
Description [p.13]	{bindings [p.13] }, {element declarations [p.13] }, {interfaces [p.13] }, {services [p.13] }, {type definitions [p.13] }
Element Declaration [p.12]	{name [p.12] }, {system [p.12] }
Endpoint [p.55]	{address [p.55] }, {binding [p.55] }, {name [p.55] }, {parent [p.55] }
Interface [p.17]	{extended interfaces [p.18] }, {interface faults [p.18] }, {interface operations [p.18] }, {name [p.17] }
Interface Fault [p.21]	{element declaration [p.21] }, {message content model [p.21] }, {name [p.21] }, {parent [p.21] }
Interface Fault Reference [p.35]	{direction [p.35] }, {interface fault [p.35] }, {message label [p.35] }, {parent [p.35] }
Interface Message Reference [p.30]	{direction [p.31] }, {element declaration [p.31] }, {message content model [p.31] }, {message label [p.30] }, {parent [p.31] }

Interface Operation [p.25]	{interface fault references [p.25] }, {interface message references [p.25] }, {message exchange pattern [p.25] }, {name [p.25] }, {parent [p.25] }, {style [p.25] }
Service [p.52]	{endpoints [p.52] }, {interface [p.52] }, {name [p.52] }
Type Definition [p.12]	{name [p.12] }, {system [p.12] }
Property	**Where Defined**
address	Endpoint.{address [p.55] }
binding	Endpoint.{binding [p.55] }
binding fault references	Binding Operation.{binding fault references [p.44] }
binding faults	Binding.{binding faults [p.39] }
binding message references	Binding Operation.{binding message references [p.44] }
binding operations	Binding.{binding operations [p.39] }
bindings	Description.{bindings [p.13] }
direction	Interface Fault Reference.{direction [p.35] }, Interface Message Reference.{direction [p.31] }
element declaration	Interface Fault.{element declaration [p.21] }, Interface Message Reference.{element declaration [p.31] }
element declarations	Description.{element declarations [p.13] }
endpoints	Service.{endpoints [p.52] }
extended interfaces	Interface.{extended interfaces [p.18] }
interface	Binding.{interface [p.39] }, Service.{interface [p.52] }
interface fault	Binding Fault.{interface fault [p.42] }, Interface Fault Reference.{interface fault [p.35] }
interface fault reference	Binding Fault Reference.{interface fault reference [p.49] }
interface fault references	Interface Operation.{interface fault references [p.25] }
interface faults	Interface.{interface faults [p.18] }

interface message reference	Binding Message Reference.{interface message reference [p.47] }
interface message references	Interface Operation.{interface message references [p.25] }
interface operation	Binding Operation.{interface operation [p.44] }
interface operations	Interface.{interface operations [p.18] }
interfaces	Description.{interfaces [p.13] }
message content model	Interface Fault.{message content model [p.21] }, Interface Message Reference.{message content model [p.31] }
message exchange pattern	Interface Operation.{message exchange pattern [p.25] }
message label	Interface Fault Reference.{message label [p.35] }, Interface Message Reference.{message label [p.30] }
name	.{name [p.58] }, Binding.{name [p.39] }, Element Declaration.{name [p.12] }, Endpoint.{name [p.55] }, Interface.{name [p.17] }, Interface Fault.{name [p.21] }, Interface Operation.{name [p.25] }, Service.{name [p.52] }, Type Definition.{name [p.12] }
parent	.{parent [p.13] }, Binding Fault.{parent [p.42] }, Binding Fault Reference.{parent [p.49] }, Binding Message Reference.{parent [p.47] }, Binding Operation.{parent [p.44] }, Endpoint.{parent [p.55] }, Interface Fault.{parent [p.21] }, Interface Fault Reference.{parent [p.35] }, Interface Message Reference.{parent [p.31] }, Interface Operation.{parent [p.25] }
services	Description.{services [p.13] }
style	Interface Operation.{style [p.25] }
system	Element Declaration.{system [p.12] }, Type Definition.{system [p.12] }
type	Binding.{type [p.39] }
type definitions	Description.{type definitions [p.13] }

E. Assertion Summary (Non-Normative)

This appendix summarizes assertions about WSDL 2.0 documents and components that are not enforced by the WSDL 2.0 schema. Each assertion is assigned a unique identifier which WSDL 2.0 processors may use to report errors.

Table E-1. Summary of Assertions about WSDL 2.0 Documents

Id	Assertion
Description-1004 [p.14]	If a WSDL 2.0 document is split into multiple WSDL 2.0 documents (which may be combined as needed via **4.1 Including Descriptions** [p.67]), then the `targetNamespace` *attribute information item* SHOULD resolve to a master WSDL 2.0 document that includes all the WSDL 2.0 documents needed for that service description.
Description-1005 [p.14]	Zero or more *element information item*s amongst its [children], in order as follows:
Description-1006 [p.15]	Its value MUST be an absolute IRI (see [*IETF RFC 3987 [p.78]*]) and should be dereferencable.
Import-1082 [p.69]	As with XML schema, any WSDL 2.0 document that references a foreign component MUST have a `wsdl:import` *element information item* for the associated foreign namespace (but which does not necessarily provide a `location` *attribute information item* that identifies the WSDL 2.0 document in which the referenced component is defined).
Import-1083 [p.69]	If a WSDL 2.0 document contains more than one `wsdl:import` *element information item* for a given value of the `namespace` *attribute information item*, then they MUST provide different values for the `location` *attribute information item.*
Import-1084 [p.70]	This value MUST NOT match the actual value of `targetNamespace` *attribute information item* in the enclosing WSDL 2.0 document.
Import-1085 [p.70]	If the location attribute in the `import` *element information item* is dereferencable, then it MUST reference a WSDL 2.0 document.
Import-1086 [p.70]	If the `location` *attribute information item* of the `import` *element information item* is dereferencable, then the actual value of the `namespace` *attribute information item* MUST be identical to the actual value of the `targetNamespace` *attribute information item* of the referenced WSDL 2.0 document (see **7. Locating WSDL 2.0 Documents** [p.75]).
Include-1080 [p.68]	The IRI indicated by `location` MUST resolve to a WSDL 2.0 document.
Include-1081 [p.68]	The actual value of the `targetNamespace` *attribute information item* of the included WSDL 2.0 document MUST match the actual value of the `targetNamespace` *attribute information item* of the `description` *element information item* which is the [parent] of the `include` *element information item.*

Interface-1012 [p.20]	Its value, if present, MUST contain absolute IRIs (see [*IETF RFC 3987 [p.78]*]).
InterfaceFault-1017 [p.24]	If the `element` *attribute information item* has a value, then it MUST resolve to an Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component.
InterfaceFaultReference-1040 [p.37]	The `messageLabel` *attribute information item* MUST be present in the XML representation of an Interface Fault Reference [p.35] component with a given {direction [p.35] }, if the {message exchange pattern [p.25] } of the parent Interface Operation [p.25] component has more than one fault with that direction.
InterfaceMessageReference-1036 [p.34]	If the `element` *attribute information item* has a value, then it MUST resolve to an Element Declaration [p.12] component from the {element declarations [p.13] } property of the Description [p.13] component.
Location-1092 [p.75]	It MUST NOT appear on a `wsdl:description` element or any of its children/descendants.
Location-1094 [p.75]	For each pair of IRIs, if the location IRI of the pair is dereferencable, then it MUST reference a WSDL 2.0 (or 1.1) document whose target namespace is the namespace IRI of the pair.
MessageLabel-1030 [p.33]	If the `messageLabel` *attribute information item* of an interface message reference *element information item* is present, then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1031 [p.33]	If the `messageLabel` *attribute information item* of an interface message reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1032 [p.33]	If the local name is `input` then the message exchange pattern MUST have at least one placeholder message with direction "In".
MessageLabel-1033 [p.33]	If the local name is `output` then the message exchange pattern MUST have at least one placeholder message with direction "Out".
MessageLabel-1034 [p.33]	If the local name is `infault` then the message exchange pattern MUST support at least one fault in the "In" direction.
MessageLabel-1035 [p.33]	If the local name is `outfault` then the message exchange pattern MUST support at least one fault in the "Out" direction.

MessageLabel-1041 [p.37]	The `messageLabel` *attribute information item* of an interface fault reference *element information item* MUST be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1042 [p.37]	If the `messageLabel` *attribute information item* of an interface fault reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1043 [p.37]	If the `messageLabel` *attribute information item* of an interface fault reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1053 [p.48]	If the `messageLabel` *attribute information item* of a binding message reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1054 [p.48]	If the `messageLabel` *attribute information item* of a binding message reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1056 [p.51]	The `messageLabel` *attribute information item* of a binding fault reference *element information item* MUST be present if the message exchange pattern has more than one placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1057 [p.51]	If the `messageLabel` *attribute information item* of a binding fault reference *element information item* is present then its actual value MUST match the {message label [p.30] } of some placeholder message with {direction [p.31] } equal to the message direction.
MessageLabel-1058 [p.51]	If the `messageLabel` *attribute information item* of a binding fault reference *element information item* is absent then there MUST be a unique placeholder message with {direction [p.31] } equal to the message direction.
QName-resolution-1064 [p.59]	A Description [p.13] component MUST NOT have such broken references.

Schema-1066 [p.61]	A WSDL 2.0 document MUST NOT refer to XML Schema components in a given namespace UNLESS an `xs:import` or `xs:schema` *element information item* for that namespace is present OR the namespace is the XML Schema namespace, http://www.w3.org/2001/XMLSchema, which contains built-in types as defined in XML Schema Part 2: Datatypes Second Edition [*XML Schema: Datatypes [p.79]*].
Schema-1069 [p.62]	The referenced schema MUST contain a `targetNamespace` *attribute information item* on its `xs:schema` *element information item*.
Schema-1070 [p.62]	The value of the `targetNamespace` *attribute information item* of the `xs:schema` *element information item* of an imported schema MUST equal the value of the `namespace` of the `import` *element information item* in the importing WSDL 2.0 document.
Schema-1073 [p.64]	A WSDL 2.0 document MUST NOT define the same element or type in more than one inlined schema.
Schema-1075 [p.65]	A specification of extension syntax for an alternative schema language MUST use a namespace that is different than the namespace of XML Schema.
Schema-1076 [p.65]	The namespace used for an alternate schema language MUST be an absolute IRI.
Schema-1079 [p.66]	If `wsdlx:interface` and `wsdlx:binding` are used together then they MUST satisfy the same consistency rules that apply to the {interface [p.52] } property of a Service [p.52] component and the {binding [p.55] } property of a nested Endpoint [p.55] component, that is either the binding refers the interface of the service or the binding refers to no interface.
Types-1074 [p.65]	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*).
Types-1077 [p.66]	The type of the `wsdlx:interface` *attribute information item* is an *xs:QName* that specifies the {name [p.17] } property of an Interface [p.17] component.
Types-1078 [p.66]	The type of the `wsdlx:binding` *attribute information item* is an *xs:QName* that specifies the {name [p.39] } property of a Binding [p.39] component.

Table E-2. Summary of Assertions about WSDL 2.0 Components

Id	Assertion
Binding-1044 [p.38]	If a Binding [p.39] component specifies any operation-specific binding details (by including Binding Operation [p.44] components) or any fault binding details (by including Binding Fault [p.42] components), then it MUST specify an interface the Binding [p.39] component applies to, so as to indicate which interface the operations come from.
Binding-1045 [p.39]	A Binding [p.39] component that defines bindings for an Interface [p.17] component MUST define bindings for all the operations of that Interface [p.17] component.
Binding-1046 [p.39]	Similarly, whenever a reusable Binding [p.39] component (i.e. one that does not specify an Interface [p.17] component) is applied to a specific Interface [p.17] component in the context of an Endpoint [p.55] component (see **2.13.1 The Endpoint Component** [p.54]), the Binding [p.39] component MUST define bindings for each Interface Operation [p.25] and Interface Fault [p.21] component of the Interface [p.17] component, via a combination of properties defined on the Binding [p.39] component itself and default binding rules specific to its binding type.
Binding-1047 [p.39]	A Binding [p.39] component that defines bindings for an Interface [p.17] component MUST define bindings for all the faults of that Interface [p.17] component that are referenced from any of the operations in that Interface [p.17] component.
Binding-1048 [p.39]	This *xs:anyURI* MUST be an absolute IRI as defined by [*IETF RFC 3987 [p.78]*].
Binding-1049 [p.39]	For each Binding [p.39] component in the {bindings [p.13] } property of a Description [p.13] component, the {name [p.39] } property MUST be unique.
BindingFault-1050 [p.42]	For each Binding Fault [p.42] component in the {binding faults [p.39] } property of a Binding [p.39] component, the {interface fault [p.42] } property MUST be unique.
BindingFaultReference-1055 [p.49]	For each Binding Fault Reference [p.49] component in the {binding fault references [p.44] } property of a Binding Operation [p.44] component, the {interface fault reference [p.49] } property MUST be unique.

BindingFaultReference-1059 [p.52]	There MUST be an Interface Fault Reference [p.35] component in the {interface fault references [p.25] } of the Interface Operation [p.25] being bound with {message label [p.35] } equal to the effective message label and with {interface fault [p.35] } equal to an Interface Fault [p.21] component with {name [p.21] } equal to the actual value of the `ref` *attribute information item*.
BindingMessageReference-1052 [p.47]	For each Binding Message Reference [p.47] component in the {binding message references [p.44] } property of a Binding Operation [p.44] component, the {interface message reference [p.47] } property MUST be unique.
BindingOperation-1051 [p.44]	For each Binding Operation [p.44] component in the {binding operations [p.39] } property of a Binding [p.39] component, the {interface operation [p.44] } property MUST be unique.
CanonFragId-1097 [p.89]	The fragment identifier consists of a sequence zero or more `xmlns()` pointer parts followed by exactly one `wsdl.*()` pointer part.
CanonFragId-1098 [p.89]	Each `xmlns()` pointer part that appears in the fragment identifier defines a namespace that is referenced by the `wsdl.*()` pointer part.
CanonFragId-1099 [p.89]	Each `xmlns()` pointer part defines a unique namespace.
CanonFragId-1100 [p.89]	The `xmlns()` pointer parts define namespaces in the same order as they are referenced in the `wsdl.*()` pointer part.
CanonFragId-1101 [p.89]	The namespace prefixes defined by the `xmlns()` pointer parts are named `ns1` , `ns2` , etc., in the order of their appearance.
CanonFragId-1102 [p.89]	The fragment identifier contains no optional whitespace.
CanonFragId-1103 [p.89]	No `xmlns()` pointer part defines a namespace for the targetNamespace of the WSDL 2.0 document.
Compare-URI-IRI-1065 [p.59]	When such absolute URIs and IRIs are being compared to determine equivalence (see **2.15 Equivalence of Components** [p.58]), they MUST be compared character-by-character as indicated in [*IETF RFC 3987 [p.78]*].
Description-1001 [p.14]	The value of the `targetNamespace` *attribute information item* SHOULD be dereferencable.
Description-1002 [p.14]	It SHOULD resolve to a human or machine processable document that directly or indirectly defines the intended semantics of those components.

Description-1003 [p.14]	It MAY resolve to a WSDL 2.0 document that provides service description information for that namespace.
Description-1067 [p.62]	For each component in the imported namespace, a corresponding Element Declaration [p.12] component or Type Definition [p.12] component MUST appear in the {element declarations [p.13] } or {type definitions [p.13] } property respectively of the Description [p.13] component corresponding to the WSDL document that imports the schema, or that imports directly or indirectly a WSDL document that imports the schema.
Description-1068 [p.62]	Schema components not in an imported namespace MUST NOT appear in the {element declarations [p.13] } or {type definitions [p.13] } properties.
Description-1071 [p.63]	For each component defined and declared in the inlined schema document or included by `xs:include`, a corresponding Element Declaration [p.12] component or Type Definition [p.12] component MUST appear in the {element declarations [p.13] } property or {type definitions [p.13] } property respectively of the Description [p.13] component corresponding to the WSDL document that contains the schema, or that imports directly or indirectly a WSDL document that contains the schema.
Description-1072 [p.63]	Schema components not defined or declared in the inlined schema document or included by `xs:include` MUST NOT appear in the {element declarations [p.13] } or {type definitions [p.13] } properties.
Endpoint-1061 [p.55]	This *xs:anyURI* MUST be an absolute IRI as defined by [*IETF RFC 3987 [p.78]*].
Endpoint-1062 [p.55]	For each Endpoint [p.55] component in the {endpoints [p.52] } property of a Service [p.52] component, the {binding [p.55] } property MUST either be a Binding [p.39] component with an unspecified {interface [p.39] } property or a Binding [p.39] component with an {interface [p.39] } property equal to the {interface [p.52] } property of the Service [p.52] component.
Equivalence-1063 [p.58]	Extension properties which are not string values, sets of strings or references MUST describe their values' equivalence rules.
Extensibility-1089 [p.72]	An extension that is NOT marked as mandatory MUST NOT invalidate the meaning of any part of a WSDL 2.0 document.

Extensibility-1090 [p.73]	If a WSDL 2.0 document declares an extension as optional (i.e., NON-mandatory), then the Web service MUST NOT assume that the client supports that extension *unless* the Web service knows (through some other means) that the client has in fact elected to engage and support that extension.
Extensibility-1091 [p.73]	Therefore, the Web service MUST support every extension that is declared as optional in the WSDL 2.0 document, in addition to supporting every extension that is declared as mandatory.
Extension-1088 [p.72]	The meaning of an extension SHOULD be defined (directly or indirectly) in a document that is available at its namespace IRI.
FragId-1095 [p.82]	For QNames, any prefix MUST be defined by a preceding xmlns pointer part.
FragId-1096 [p.82]	The fragment identifier in a WSDL 2.0 component IRI-reference MUST resolve to some component as defined by the construction rules in Table A-1 [p.82] .
ImportInclude-1087 [p.71]	The semantics of an extension MUST NOT depend on how components are brought into a component model instance via <import> or <include>.
Interface-1009 [p.17]	To avoid circular definitions, an interface MUST NOT appear in the set of interfaces it extends, either directly or indirectly.
Interface-1010 [p.18]	For each Interface [p.17] component in the {interfaces [p.13] } property of a Description [p.13] component, the {name [p.17] } property MUST be unique.
Interface-1011 [p.19]	The list of *xs:QName* in an `extends` *attribute information item* MUST NOT contain duplicates.
InterfaceFault-1013 [p.21]	An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.
InterfaceFault-1014 [p.21]	When the {message content model [p.21] } property has the value *#any* or *#none* the {element declaration [p.21] } property MUST be empty.
InterfaceFault-1015 [p.22]	In cases where, due to an interface extending one or more other interfaces, two or more Interface Fault [p.21] components have the same value for their {name [p.21] } property, then the component models of those Interface Fault [p.21] components MUST be equivalent (see **2.15 Equivalence of Components** [p.58]).

InterfaceFault-1016 [p.22]	For the above reason, it is considered good practice to ensure, where necessary, that the local name of the {name [p.21] } property of Interface Fault [p.21] components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error.
InterfaceFaultReference-1037 [p.35]	The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
InterfaceFaultReference-1038 [p.35]	The direction MUST be consistent with the direction implied by the fault propagation ruleset used in the message exchange pattern of the operation.
InterfaceFaultReference-1039 [p.35]	For each Interface Fault Reference [p.35] component in the {interface fault references [p.25] } property of an Interface Operation [p.25] component, the combination of its {interface fault [p.35] } and {message label [p.35] } properties MUST be unique.
InterfaceMessageReference-1025 [p.31]	An *xs:token* with one of the values *in* or *out*, indicating whether the message is coming to the service or going from the service, respectively.
InterfaceMessageReference-1026 [p.31]	The direction MUST be the same as the direction of the message identified by the {message label [p.30] } property in the {message exchange pattern [p.25] } of the Interface Operation [p.25] component this is contained within.
InterfaceMessageReference-1027 [p.31]	An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.
InterfaceMessageReference-1028 [p.31]	When the {message content model [p.31] } property has the value *#any* or *#none*, the {element declaration [p.31] } property MUST be empty.
InterfaceMessageReference-1029 [p.31]	For each Interface Message Reference [p.30] component in the {interface message references [p.25] } property of an Interface Operation [p.25] component, its {message label [p.30] } property MUST be unique.
InterfaceOperation-1018 [p.25]	This *xs:anyURI* MUST be an absolute IRI (see [*IETF RFC 3987 [p.78]*]).
InterfaceOperation-1019 [p.25]	These *xs:anyURI*s MUST be absolute IRIs (see [*IETF RFC 3986 [p.78]*]).
InterfaceOperation-1020 [p.25]	In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation [p.25] components have the same value for their {name [p.25] } property, then the component models of those Interface Operation components MUST be equivalent (see **2.15 Equivalence of Components** [p.58]).

InterfaceOperation-1021 [p.26]	For the above reason, it is considered good practice to ensure, where necessary, that the {name [p.25] } property of Interface Operation [p.25] components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error.
InterfaceOperation-1023 [p.27]	An Interface Operation [p.25] component MUST satisfy the specification defined by each operation style identified by its {style [p.25] } property.
Location-1093 [p.75]	Its actual value MUST be a list of pairs of IRIs; where the first IRI of a pair, which MUST be an absolute IRI as defined in [*IETF RFC 3987 [p.78]*], indicates a WSDL 2.0 (or 1.1) namespace name, and, the second a hint as to the location of a WSDL 2.0 document defining WSDL 2.0 components (or WSDL 1.1 elements [*WSDL 1.1 [p.79]*]) for that namespace name.
MEP-1022 [p.26]	A message exchange pattern is itself uniquely identified by an absolute IRI, which is used as the value of the {message exchange pattern [p.25] } property of the Interface Operation [p.25] component, and which specifies the fault propagation ruleset that its faults obey.
MessageLabel-1024 [p.30]	The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
Service-1060 [p.52]	For each Service [p.52] component in the {services [p.13] } property of a Description [p.13] component, the {name [p.52] } property MUST be unique.
Types-1007 [p.16]	Each XML Schema element declaration MUST have a unique QName.
Types-1008 [p.17]	Each XML Schema type definition MUST have a unique QName.