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

Authors and Contributors:

Roberto Chinnici (Sun Microsystems)
Jean-Jacques Moreau (Canon)
Arthur Ryman (IBM)
Sanjiva Weerawarana (WSO2)

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

This page is intentionally left blank.

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] describes extensions for message exchange patterns, operation safety, operation styles and binding extensions (for SOAP [SOAP 1.2 Part 1: Messaging Framework (Second Edition)] and HTTP [IETF RFC 2616]).

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 on page 74, 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]) 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] 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 on page 5 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].

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].

1.4.3. XML Schema anyURI

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

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]).

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* on page 76.
wsdlx	http://www.w3.org/ns/wsdl-extensions	Defined by this specification § 3.3 – Describing Messages that Refer to Services and Endpoints on page 68.
wrpc	http://www.w3.org/ns/wsdl/rpc	Defined by WSDL 2.0: Adjuncts [WSDL 2.0 Adjuncts].
wsoap	http://www.w3.org/ns/wsdl/soap	Defined by WSDL 2.0: Adjuncts [WSDL 2.0 Adjuncts].
whttp	http://www.w3.org/ns/wsdl/http	Defined by WSDL 2.0: Adjuncts [WSDL 2.0 Adjuncts].
xs	http://www.w3.org/2001/XMLSchema	Defined in the W3C XML Schema specification [XML Schema: Structures], [XML Schema: Datatypes].
xsi	http://www.w3.org/2001/XMLSchema-instance	Defined in the W3C XML Schema specification [XML Schema: Structures], [XML Schema: Datatypes].

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], 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 on page 66.

1.4.6. XML Information Set Properties

This specification refers to properties in the XML Information Set [XML Information Set]. 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 on page 5. Such properties are denoted by curly brackets, e.g. name, interfaces.

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] 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].

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 on page 3. 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 on page 3. 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 Appendix E – Assertion Summary on page 100.

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].

A WSDL 2.0 document, and its related documents, defines a set of components that together form an instance of a Component Model. This specification defines the structure and constraints on the components in a valid component model instance.

Let ComponentModel be the set of valid component model instances:

DescriptionCM ElementDeclarationCM TypeDefinitionCM InterfaceCM InterfaceFaultCM InterfaceOperationCM InterfaceMessageReferenceCM InterfaceFaultReferenceCM BindingCM BindingFaultCM BindingOperationCM BindingMessageReferenceCM BindingFaultReferenceCM ServiceCM EndpointCM
The definition of ComponentModel is built up from definitions for each of the component types. A component model instance is valid if and only if the constraints on each of the component types are satisfied. The component type definitions are given in the following sections.

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].

Let Component be the union of each of the component types that appear in the WSDL 2.0 component model:

Component ::= descriptionDescription | elementDeclElementDeclaration | typeDefTypeDefinition | interfaceInterface | interfaceFaultInterfaceFault | interfaceOpInterfaceOperation | interfaceMessageRefInterfaceMessageReference | interfaceFaultRefInterfaceFaultReference | bindingBinding | bindingFaultBindingFault | bindingOpBindingOperation | bindingMessageRefBindingMessageReference | bindingFaultRefBindingFaultReference | serviceService | endpointEndpoint
The Component type is an example of a Z Notation *free type*. The structure of a free type is similar to that of a variant record or discriminated union datatype that are found in some common programming languages. Each of the members of this union is formally defined in the following sections.

When a component property is said to contain another component or a set of other components, the intended meaning is that the component property contains a reference to another component or a set of references to other components. Every component contains an unique identifier that is used to express references.

Let ID be the set of all component identifier values:

[ID]
The ID type is an example of a Z Notation *basic set*. The structure of a basic set is immaterial. The only relevant aspect of ID is that it contains enough members to uniquely identify each component, and that these identifiers can be compared for equality. These identifiers are similar to XML element ids or object identifiers that are found in common object-oriented programming languages.

Every component has an identifier which uniquely identifies it within a component model instance.

Let Identifier be the set of component identifier properties:

- Let id be the identifier of the component.

id : ID
The Identifier set is a an example of Z Notation *schema*. The structure of a Z schema is similar to that of a record or struct datatype that are found in many common programming languages. The fields of an instance of a Z schema are selected using the usual dot notation, e.g. x.id selects the id field of the instance x.

All component properties that contain an ID, except for Identifier, refer to other components. Every ID value that appears in a component reference corresponds to a unique component in the component model with that identifier.

Let Id map components to their identifiers:

Id : Component ID x : Description @ Id(description(x)) = x.id x : ElementDeclaration @ Id(elementDecl(x)) = x.id x : TypeDefinition @ Id(typeDef(x)) = x.id x : Interface @ Id(interface(x)) = x.id x : InterfaceFault @ Id(interfaceFault(x)) = x.id x : InterfaceOperation @ Id(interfaceOp(x)) = x.id x : InterfaceMessageReference @ Id(interfaceMessageRef(x)) = x.id x : InterfaceFaultReference @ Id(interfaceFaultRef(x)) = x.id x : Binding @ Id(binding(x)) = x.id x : BindingFault @ Id(bindingFault(x)) = x.id x : BindingOperation @ Id(bindingOp(x)) = x.id x : BindingMessageReference @ Id(bindingMessageRef(x)) = x.id x : BindingFaultReference @ Id(bindingFaultRef(x)) = x.id x : Service @ Id(service(x)) = x.id x : Endpoint @ Id(endpoint(x)) = x.id
The Id function is an example of a Z Notation *axiomatic definition*. An axiomatic definition declares an object and then characterizes it with a set of axioms or logical constraints that it satisfies. In this case, the Id function is constrained by giving its value on each possible type of component, which uniquely defines it.

A component model is a set of uniquely identified components that satisfy a set of validity constraints which are described in the following sections.

Let ComponentModel1 be the base set of component models. This set will be further constrained in the following sections:

- Let components be the set of components in the component model.
- Let componentIds be the set of identifiers of components in the component model.

components : Component componentIds : ID x, y : components @ Id(x) = Id(y) x = y componentIds = {~x : components @ Id(x)~}

- No two components have the same identifier.

An identifier is valid if it is the identifier of a component in the component model.

Let IdentifierValid express this validity constraint:

ComponentModel1 Identifier id componentIds
In order to express the additional constraints on the component model, it is convenient to define the subsets of components of each type and their corresponding subsets of identifiers.

Let InterfaceComponents define the subsets of components that are related to the Interface component:

- Let interfaceComps be the subset of Interface components.
- Let interfaceFaultComps be the subset of Interface Fault components.

- Let interfaceOpComps be the subset of Interface Operation components.
- Let interfaceMessageRefComps be the subset of Interface Message Reference components.
- Let interfaceFaultRefComps be the subset of Interface Fault Reference components.

ComponentModel1 interfaceComps : Interface interfaceFaultComps : InterfaceFault interfaceOpComps : InterfaceOperation interfaceMessageRefComps : InterfaceMessageReference interfaceFaultRefComps : InterfaceFaultReference interfaceComps = {~x : Interface | interface(x) components~} interfaceFaultComps = {~x : InterfaceFault | interfaceFault(x) components~} interfaceOpComps = {~x : InterfaceOperation | interfaceOp(x) components~} interfaceMessageRefComps = {~x : InterfaceMessageReference | interfaceMessageRef(x) components~} interfaceFaultRefComps = {~x : InterfaceFaultReference | interfaceFaultRef(x) components~}
The definition of InterfaceComponents is an example of Z Notation *schema inclusion*. In Z schema inclusion all the fields and constraints of the included Z schema, e.g. ComponentModel1 are added to the including Z schema, e.g. InterfaceComponents.

Let InterfaceComponentIds define the subsets of component identifiers that are related to the Interface component:

- Let interfaceIds be the subset of Interface component identifiers.
- Let interfaceFaultIds be the subset of Interface Fault component identifiers.
- Let interfaceOpIds be the subset of Interface Operation component identifiers.
- Let interfaceMessageRefIds be the subset of Interface Message Reference component identifiers.
- Let interfaceFaultRefIds be the subset of Interface Fault Reference component identifiers.

InterfaceComponents interfaceIds : ID interfaceFaultIds : ID interfaceOpIds : ID interfaceMessageRefIds : ID interfaceFaultRefIds : ID interfaceIds = {~x : interfaceComps @ x.id~} interfaceFaultIds = {~x : interfaceFaultComps @ x.id~} interfaceOpIds = {~x : interfaceOpComps @ x.id~} interfaceMessageRefIds = {~x : interfaceMessageRefComps @ x.id~} interfaceFaultRefIds = {~x : interfaceFaultRefComps @ x.id~}
Let BindingComponents define the subsets of components that are related to the Binding component:

- Let bindingComps be the subset of Binding components.
- Let bindingFaultComps be the subset of Binding Fault components.
- Let bindingOpComps be the subset of Binding Operation components.
- Let bindingMessageRefComps be the subset of Binding Message Reference components.
- Let bindingFaultRefComps be the subset of Binding Fault Reference components.

ComponentModel1 bindingComps : Binding bindingFaultComps : BindingFault bindingOpComps : BindingOperation bindingMessageRefComps : BindingMessageReference bindingFaultRefComps : BindingFaultReference bindingComps = {~x : Binding | binding(x) components~} bindingFaultComps = {~x : BindingFault | bindingFault(x) components~} bindingOpComps = {~x : BindingOperation | bindingOp(x) components~} bindingMessageRefComps = {~x : BindingMessageReference | bindingMessageRef(x) components~} bindingFaultRefComps = {~x : BindingFaultReference | bindingFaultRef(x) components~}
Let BindingComponentIds define the subsets of component identifiers that are related to the Binding component:

- Let bindingIds be the subset of Binding component identifiers.

- Let bindingFaultIds be the subset of Binding Fault component identifiers.
- Let bindingOpIds be the subset of Binding Operation component identifiers.
- Let bindingMessageRefIds be the subset of Binding Message Reference component identifiers.
- Let bindingFaultRefIds be the subset of Binding Fault Reference component identifiers.

BindingComponents bindingIds : ID bindingFaultIds : ID bindingOpIds : ID bindingMessageRefIds : ID bindingFaultRefIds : ID bindingIds = {~x : bindingComps @ x.id~} bindingFaultIds = {~x : bindingFaultComps @ x.id~} bindingOpIds = {~x : bindingOpComps @ x.id~} bindingMessageRefIds = {~x : bindingMessageRefComps @ x.id~} bindingFaultRefIds = {~x : bindingFaultRefComps @ x.id~}
Let ServiceComponents define the subsets of components that are related to the Service component:

- Let serviceComps be the subset of Service components.
- Let endpointComps be the subset of Endpoint components.

ComponentModel1 serviceComps : Service endpointComps : Endpoint serviceComps = {~x : Service | service(x) components~} endpointComps = {~x : Endpoint | endpoint(x) components~}
Let ServiceComponentIds define the subsets of component identifiers that are related to the Service component:

- Let serviceIds be the subset of Service component identifiers.
- Let endpointIds be the subset of Endpoint component identifiers.

ServiceComponents serviceIds : ID endpointIds : ID serviceIds = {~x : serviceComps @ x.id~} endpointIds = {~x : endpointComps @ x.id~}
Let OtherComponents define the subsets of the other component types:

- Let descriptionComps be the subset of Description components.
- Let elementDeclComps be the subset of Element Declaration components.
- Let typeDefComps be the subset of Type Definition components.

ComponentModel1 descriptionComps : Description elementDeclComps : ElementDeclaration typeDefComps : TypeDefinition descriptionComps = {~x : Description | description(x) components~} elementDeclComps = {~x : ElementDeclaration | elementDecl(x) components~} typeDefComps = {~x : TypeDefinition | typeDef(x) components~}
Let OtherComponentIds define the subsets of other component identifiers:

- Let descriptionIds be the subset of Description component identifiers.
- Let elementDeclIds be the subset of Element Declaration component identifiers.
- Let typeDefIds be the subset of Type Definition component identifiers.

OtherComponents descriptionIds : ID elementDeclIds : ID typeDefIds : ID descriptionIds = {~x : descriptionComps @ x.id~} elementDeclIds = {~x : elementDeclComps @ x.id~} typeDefIds = {~x : typeDefComps @ x.id~}
Let ComponentModel2 be the basic component model, augmented with the definitions of the subsets of each component type and their corresponding identifiers:

ComponentModel2 InterfaceComponentIds BindingComponentIds ServiceComponentIds OtherComponentIds
The definition of ComponentModel2 is an example of Z Notation *schema conjunction*. In Z schema conjunction, the resulting Z schema, e.g. ComponentModel2, contains all the fields of the conjoined Z schemas,

e.g. InterfaceComponentIds, BindingComponentIds, ServiceComponentIds, and OtherComponentIds, and its constraint is the conjunction (logical and) of their constraints.

The component types in the component model have an identifier. It is convenient to put this field into a base Z schema that can be included in other component schemas.

Let Base be the common base Z schema for all component types that have an identifier:

Identifier
The base properties of a component are valid when the identifiers are valid:

Let BaseValid be this validity constraint on the base fields of a component:

IdentifierValid
Nested components have an additional parent property.

Let NestedBase be the common base schema for all nested component types:

Base Parent
The properties of a nested base component are valid when the base properties are valid and the parent property is valid.

Let NestedBaseValid be the validity constraints for nested components:

BaseValid ParentValid
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.

An OPTIONAL simple property type is treated as a set-valued type that contains at most one member. If the property is absent then its value is the empty set. If the property is present then its value is the singleton set that contains the actual value of the property.

Let OPTIONAL[X] be the OPTIONAL values of type X where X is a property type:

OPTIONAL : (X) OPTIONAL = {} {~x : X @ {x}~}

- An optional value of type X is either the empty set or a singleton set that contains one member of X.

For example, OPTIONAL[{True, False}] = {, {True}, {False}}.

The definition of OPTIONAL is an example of Z Notation *generic definition*. A Z generic definition defines an object whose type depends on the types of one or more sets that are given as arguments to the definition. A Z generic definition is similar to a generic, template, or parameterized type that are found in common programming languages.

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 on page 61) of the simple types defined by the XML Schema 1.0 specification [XML Schema: Datatypes].

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 on page 69.

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 on page 69.

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 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], 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 on page 73. 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.

Let ElementContentModel be the set of all models that define the allowable values for the [children] and [attribute] properties of an *element information item*:

[ElementContentModel]
The detailed structure of ElementContentModel is immaterial for the purposes of this specification. It is can be safely thought of as some superset of the set of all XML Schema complex type definitions.

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.

Let ElementDeclaration be the type of Element Declaration components:

- Let name be the QName defined by the [local name] and [namespace name] properties of the *element information item*.
- Let system be the namespace IRI of the type system.

- Let elementContentModel be the element content model that constrains the allowable contents of the [children] and [attribute] properties of the *element information item*.

Identifier name : QName system : AbsoluteURI elementContentModel : ElementContentModel
Each Element Declaration component is uniquely identified by the combination of its name and system properties within the component model.

Let ElementDeclarationCM express this constraint:

ComponentModel2 x, y : elementDeclComps | x.name = y.name x.system = y.system @ x = y

- No two Element Declaration components have the same name and system properties.

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.

Let TypeDefinition be the type of the Type Definition component:

- Let name be the QName of the type definition.
- Let system be the namespace IRI of the type system.
- Let elementContentModel be the element content model that constrains the allowable contents of the [children] and [attribute] properties of the *element information item* described by the type definition.

Identifier name : QName system : AbsoluteURI elementContentModel : ElementContentModel
Each Type Definition component is uniquely identified by the combination of its name and system properties within the component model.

Let TypeDefinitionCM express this constraint:

ComponentModel2 x, y : typeDefComps | x.name = y.name x.system = y.system @ x = y

- No two Type Definition components have the same name and system properties.

Interface, Binding, Service, Element Declaration, and Type Definition components are directly contained in the Description component and are referred to as *top-level components*. The top-level WSDL 2.0 components contain other components, e.g. Interface Operation and Endpoint, 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.

Let TopLevelComponent be the set of all top-level components:

TopLevelComponent == elementDecl typeDef interface binding service
Let Name map a top-level component to its QName name property:

Name : TopLevelComponent QName x : ElementDeclaration @ Name(elementDecl(x)) = x.name x : TypeDefinition @ Name(typeDef(x)) = x.name x : Interface @ Name(interface(x)) = x.name x : Binding @ Name(binding(x)) = x.name x : Service @ Name(service(x)) = x.name
Let Parent represent the parent property of a nested component:

Identifier parent : ID
The parent of a nested component in the component model MUST also be in the component model. No component is its own parent.

Let ParentValid represent these validity constraints:

ComponentModel1 Parent parent componentIds parent id
Let NestedComponent be the set of all nested components:

NestedComponent == interfaceFault interfaceOp interfaceMessageRef interfaceFaultRef bindingFault bindingOp bindingMessageRef bindingFaultRef endpoint
Let ParentId map a nested component to its parent component identifier:

ParentId : NestedComponent ID x : InterfaceFault @ ParentId(interfaceFault(x)) = x.parent x : InterfaceOperation @ ParentId(interfaceOp(x)) = x.parent x : InterfaceMessageReference @ ParentId(interfaceMessageRef(x)) = x.parent x : InterfaceFaultReference @ ParentId(interfaceFaultRef(x)) = x.parent x : BindingFault @ ParentId(bindingFault(x)) = x.parent x : BindingOperation @ ParentId(bindingOp(x)) = x.parent x : BindingMessageReference @ ParentId(bindingMessageRef(x)) = x.parent x : BindingFaultReference @ ParentId(bindingFaultRef(x)) = x.parent x : Endpoint @ ParentId(endpoint(x)) = x.parent
The properties of the Description component are as follows:

- interfaces OPTIONAL. A set of Interface components.
- bindings OPTIONAL. A set of Binding components.
- services OPTIONAL. A set of Service components.
- element declarations OPTIONAL. A set of Element Declaration components.
- type definitions REQUIRED. A set of Type Definition components.

Let Description be the set of all Description components:

Identifier interfaces : ID bindings : ID services : ID elementDeclarations : ID typeDefinitions : ID
Let the built-in XML schema datatypes correspond to the following Type Definition components:

stringTD, booleanTD, decimalTD, floatTD, doubleTD, durationTD, dateTimeTD, timeTD, dateTD, gYearMonthTD, gYearTD, gMonthDayTD, gDayTD, gMonthTD, hexBinaryTD, base64BinaryTD, anyURITD, QNameTD, NOTATIONTD, normalizedStringTD, tokenTD, languageTD, NMTOKENTD, NMTOKENSTD, NameTD, NCNameTD, IDTD, IDREFTD, IDREFSTD, ENTITYTD, ENTITIESTD, integerTD, nonPositiveIntegerTD, negativeIntegerTD, longTD, intTD, shortTD, byteTD, nonNegativeIntegerTD, unsignedLongTD, unsignedIntTD, unsignedShortTD, unsignedByteTD, positiveIntegerTD : TypeDefinition
Let BuiltInTypeDefComps be the set of all the built-in XML schema Type Definition components:

BuiltInTypeDefComps == {stringTD, booleanTD, decimalTD, floatTD, doubleTD, durationTD, dateTimeTD, timeTD, dateTD, gYearMonthTD, gYearTD, gMonthDayTD, gDayTD, gMonthTD, hexBinaryTD, base64BinaryTD, anyURITD, QNameTD, NOTATIONTD, normalizedStringTD, tokenTD, languageTD, NMTOKENTD, NMTOKENSTD, NameTD, NCNameTD, IDTD, IDREFTD, IDREFSTD, ENTITYTD, ENTITIESTD, integerTD, nonPositiveIntegerTD, negativeIntegerTD, longTD, intTD, shortTD, byteTD, nonNegativeIntegerTD, unsignedLongTD, unsignedIntTD, unsignedShortTD, unsignedByteTD, positiveIntegerTD}
Let XMLSchemaURI be the namespace URI of XML Schema:

XMLSchemaURI : AbsoluteURI
Both the namespace name of the name property and the system property of each built-in datatypes is the XML Schema URI:

x : BuiltInTypeDefComps @ x.name.namespaceName = x.system = XMLSchemaURI

Let BuiltInTypeDefIds be the set of ids of the built-in datatypes:

BuiltInTypeDefIds == {~x : BuiltInTypeDefComps @ x.id~}
The built-in datatypes are distinct so there are forty-four ids in total:

#BuiltInTypeDefIds = 44
The XML Schema built-in datatypes are also built into WSDL 2.0.

Let DescriptionTypeDefs express this constraint on the Description:

ComponentModel2 BuiltInTypeDefComps typeDefComps
The component model contains a unique Description component.

Let DescriptionKey express this constraint on the Description component:

- Let descriptionComp be the unique Description component.

ComponentModel2 descriptionComp : Description descriptionComps = {descriptionComp}

- The component model contains a unique Description component.

Each component referred to by the properties of the Description component must exist in the component model.

Let DescriptionCM express these referential integrity constraints on the Description component:

DescriptionTypeDefs DescriptionKey descriptionComp.interfaces = interfaceIds descriptionComp.bindings = bindingIds descriptionComp.services = serviceIds descriptionComp.elementDeclarations = elementDeclIds descriptionComp.typeDefinitions = typeDefIds

- The Description component contains exactly the set of Interface components contained in the component model.
- The Description component contains exactly the set of Binding components contained in the component model.
- The Description component contains exactly the set of Service components contained in the component model.
- The Description component contains exactly the set of Element Declaration components contained in the component model.
- The Description component contains exactly the set of Type Definition components contained in the component model.

The set of top-level components contained in the Description 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 on page 70). Furthermore, all QName references, whether to the same or to different namespaces must resolve to components (see § 2.17 – QName resolution on page 63).

When using the XML Schema language to describe type system components, the inclusion of Element Declaration components and Type Definition components in a Description component is governed by the rules in § 3.1 – Using W3C XML Schema Definition Language on page 64.

In addition to WSDL 2.0 components and type system components, additional extension components MAY be added via extensibility § 6 – Language Extensibility on page 73. 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 item*s. 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 item*s § 4.2 – Importing Descriptions on page 70 that correspond to a collection with components from multiple target namespaces.

The components directly defined or included within a Description 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. It SHOULD resolve to a human or machine processable document that directly or indirectly defines the intended semantics of those components. It MAY resolve to a WSDL 2.0 document that provides service description information for that namespace.

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 on page 69), 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. 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, Binding, etc.) are in the same target namespace, then their QNames MUST be unique. However, different kinds of components (e.g., an Interface component and a Binding component) MAY have the same QName. Thus, QNames of components must be unique within the space of those components in a given target namespace.

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 item*s amongst its [attributes] as follows:
  - A REQUIRED targetNamespace *attribute information item* as described below in § 2.1.2.1 – targetNamespace *attribute information item* on page 15.

- 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], in order as follows:
  1. Zero or more documentation *element information item*s (see § 5 – Documentation on page 73).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more include *element information item*s (see § 4.1 – Including Descriptions on page 69)
     - Zero or more import *element information item*s (see § 4.2 – Importing Descriptions on page 70)
     - Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl.
  3. An OPTIONAL types *element information item* (see § 3 – Types on page 63).
  4. Zero or more *element information item*s from among the following, in any order:
     - interface *element information item*s (see § 2.2.2 – XML Representation of Interface Component on page 19).
     - binding *element information item*s (see § 2.7.2 – XML Representation of Binding Component on page 42).
     - service *element information item*s (see § 2.12.2 – XML Representation of Service Component on page 57).
     - Zero or more namespace-qualified *element information item*s 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, Binding and Service 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]) and should be dereferencable.

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 on page 14) to the properties of the Description component is described in Mapping from XML Representation to Description Component Properties Property Value interfaces The set of Interface 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 components (see). bindings The set of Binding 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 components (see). services The set of Service 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 components (see). element declarations The set of Element Declaration 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 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. type definitions The set of Type Definition 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 components. In addition, the built-in datatypes defined by XML Schema Part 2: Datatypes Second Edition , 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. .

Mapping from XML Representation to Description Component Properties

Property	Value
interfaces	The set of Interface components corresponding to all the interface *element information item*s in the [children] of the description *element information item*, if any, plus any included (via wsdl:include) or imported (via wsdl:import) Interface components (see § 4 – Modularizing WSDL 2.0 descriptions on page 69).
bindings	The set of Binding components corresponding to all the binding *element information item*s in the [children] of the description *element information item*, if any, plus any included (via wsdl:include) or imported (via wsdl:import) Binding components (see § 4 – Modularizing WSDL 2.0 descriptions on page 69).
services	The set of Service components corresponding to all the service *element information item*s in the [children] of the description *element information item*, if any, plus any included (via wsdl:include) or imported (via wsdl:import) Service components (see § 4 – Modularizing WSDL 2.0 descriptions on page 69).
element declarations	The set of Element Declaration 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 components. At a minimum this will include all the global element declarations defined by XML Schema element *element information item*s. It MAY also include any declarations from some other type system which describes the [local name], [namespace name], [attributes] and [children] properties of an *element information item*. Each XML Schema element declaration MUST have a unique QName.

type definitions	The set of Type Definition 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 components. In addition, the built-in datatypes defined by XML Schema Part 2: Datatypes Second Edition [XML Schema: Datatypes], 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.

2.2. Interface

2.2.1. The Interface Component

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

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. 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 on page 62 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 components.

Interfaces are named constructs and can be referred to by QName (see § 2.17 – QName resolution on page 63). For instance, Binding 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 components which this interface extends.
- interface faults OPTIONAL. The set of declared Interface Fault components. Note that the namespace name of the name property of each Interface Fault in this set is the same as the namespace name of the name property of this Interface component.
- interface operations OPTIONAL. A set of declared Interface Operation components. Note that the namespace name of the name property of each Interface Operation in this set is the same as the namespace name of the name property of this Interface component.

Let Interface be the set of all Interface components:

- Let allExtendedInterfaces be the set off all interfaces that are extended directly or indirectly by this interface.
- Let allInterfaceFaults be the set of all faults that are directly or indirectly on this interface.
- Let allInterfaceOperations be the set of all operations that are directly or indirectly on this interface.

Base name : QName extendedInterfaces : ID interfaceFaults : ID interfaceOperations : ID allExtendedInterfaces : ID allInterfaceFaults : ID allInterfaceOperations : ID extendedInterfaces allExtendedInterfaces interfaceFaults allInterfaceFaults interfaceOperations allInterfaceOperations
Each component referenced by an Interface component must exist in the component model.

Let InterfaceRI express the referential integrity constraints on the Interface component:

ComponentModel2 Interface | Interface interfaceComps @ BaseValid extendedInterfaces interfaceIds interfaceFaults interfaceFaultIds interfaceOperations interfaceOpIds
This Z schema introduces some additional notation. The universal quantifier Interface declares each field that is part of the Interface schema as an in-scope variable and constrains them to satisfy the rules for Interface. The expression Interface assembles these variables into Interface record or struct. The expression Interface interfaceComps constrains the Interface record to exist in the component model.

- Every Interface component satisfies the base validity constraints.
- The Interface components extended by each Interface component are contained in the component model.
- The Interface Fault components of each Interface component are contained in the component model.
- The Interface Operation components of each Interface component are contained in the component model.

For each Interface component in the interfaces property of a Description component, the name property MUST be unique.

Let InterfaceKey express the QName uniqueness constraint on the Interface component:

ComponentModel2 x, y : interfaceComps | x.name = y.name @ x = y

- No two Interface components have the same name property.

An Interface component contains nested Interface Operation and Interface Fault components. These components MUST have the Interface component as their parent.

Let InterfaceParent express the constraints on the parent properties of the nested components of an Interface component:

ComponentModel2 i : interfaceComps; if : interfaceFaultComps; io : interfaceOpComps @ if.id i.interfaceFaults if.parent = i.id io.id i.interfaceOperations io.parent = i.id

- The set of Interface Fault components contained by an Interface component is exactly the set of Interface Fault components that have that Interface component as their parent.
- The set of Interface Operation components contained by an Interface component is exactly the set of Interface Operation components that have that Interface component as their parent.

The set of all extended interfaces that are available on an Interface component consist of those that are declared on the component and those that are available on its extended interfaces.

Let InterfaceAllExtendedInterfaces express this definition:

ComponentModel2 i : interfaceComps @ i.allExtendedInterfaces = i.extendedInterfaces {~x : interfaceComps; y : ID | x.id i.extendedInterfaces y x.allExtendedInterfaces @ y~}

- An Interface component directly or indirectly extends an Interface component if it directly extends it, or if an Interface component that it directly extends, directly or indirectly extends it.

An Interface component MUST NOT directly or indirectly extend itself.

Let InterfaceExtendsAcyclic express this constraint:

ComponentModel2 i : interfaceComps @ i.id i.allExtendedInterfaces
The set of all Interface Operation components that are available on an Interface component consist of those that are contained by the Interface component and those that are available on Interface components that it directly or indirectly extends.

Let InterfaceAllInterfaceOperations express this definition:

ComponentModel2 i : interfaceComps @ i.allInterfaceOperations = i.interfaceOperations {~x : interfaceComps; y : ID | x.id i.allExtendedInterfaces y x.interfaceOperations @ y~}

- An Interface Operation component is available on an Interface component if it is contained by the Interface component or it is available on an Interface component that this Interface component directly or indirectly extends.

The set of all Interface Operation components that are available on an Interface component consist of those that are contained by the Interface component and those that are available on Interface components that it directly or indirectly extends.

Let InterfaceAllInterfaceFaults express this definition:

ComponentModel2 i : interfaceComps @ i.allInterfaceFaults = i.interfaceFaults {~x : interfaceComps; y : ID | x.id i.allExtendedInterfaces y x.interfaceFaults @ y~}

- An Interface Fault component is available on an Interface component if it is contained by the Interface component or it is available on an Interface component that this Interface component directly or indirectly extends.

Let InterfaceCM be the conjunction of all the component model constraints on Interface components.

InterfaceCM InterfaceRI InterfaceKey InterfaceParent InterfaceAllExtendedInterfaces InterfaceExtendsAcyclic InterfaceAllInterfaceOperations InterfaceAllInterfaceFaults

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 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 item*s amongst its [attributes] as follows:
  - A REQUIRED name *attribute information item* as described below in § 2.2.2.1 – name *attribute information item* with interface [owner element] on page 20.

- An OPTIONAL extends *attribute information item* as described below in § 2.2.2.2 – extends *attribute information item* on page 20.
- An OPTIONAL styleDefault *attribute information item* as described below in § 2.2.2.3 – styleDefault *attribute information item* on page 20.
- 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], in order, as follows:
  1. Zero or more documentation *element information item*s (see § 5 – Documentation on page 73).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more fault *element information item*s § 2.3.2 – XML Representation of Interface Fault Component on page 24.
     - Zero or more operation *element information item*s § 2.4.2 – XML Representation of Interface Operation Component on page 29.
     - Zero or more namespace-qualified *element information item*s 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.

2.2.2.3. styleDefault *attribute information item*

The styleDefault *attribute information item* indicates the default style (see § 2.4.1.2 – Operation Style on page 29) used to construct the element declaration properties of interface message references 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]).

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 on page 19) to the properties of the Interface component is as described in Mapping from XML Representation to Interface Component Properties Property Value name 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 The set of Interface components resolved to by the values in the extends attribute information item, if any (see). interface faults The set of Interface Fault components corresponding to the fault element information items in [children], if any. interface operations The set of Interface Operation components corresponding to the operation element information items in [children], if any. .

Mapping from XML Representation to Interface Component Properties

Property	Value
name	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	The set of Interface components resolved to by the values in the extends *attribute information item*, if any (see § 2.17 – QName resolution on page 63).
interface faults	The set of Interface Fault components corresponding to the fault *element information item*s in [children], if any.
interface operations	The set of Interface Operation components corresponding to the operation *element information item*s in [children], if any.

Recall that, per § 2.2.1 – The Interface Component on page 17, the Interface components in the extended interfaces property of a given Interface component MUST NOT contain that Interface component in any of their extended interfaces properties, that is to say, recursive extension of interfaces is disallowed.

2.3. Interface Fault

2.3.1. The Interface Fault Component

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

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

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

The Interface Fault component provides a clear mechanism to name and describe the set of faults an interface may generate. This allows operations to easily identify the individual faults they may generate

by name. This mechanism allows the ready identification of the same fault occurring across multiple operations and referenced in multiple bindings as well as reducing duplication of description for an individual fault.

Faults other than the ones described in the Interface component may also be generated at run-time, i.e. faults are an open set. The Interface 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 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 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 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*. 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 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 component in the element declarations property of the Description component. This element represents the content or “payload” of the fault. When the message content model property has the value *#any* or *#none* the element declaration property MUST be empty.
- parent REQUIRED. The Interface component that contains this component in its interface faults property.

Let InterfaceFault be the set of all Interface Fault components:

NestedBase name : QName messageContentModel : MessageContentModel elementDeclaration : OPTIONAL[ID] messageContentModel = elementToken elementDeclaration

- The message content model is *element* exactly when the element declaration property is defined.

Each component referenced by an Interface Fault component must exist in the component model.

Let InterfaceFaultRI express the referential integrity constraints on the Interface Fault component:

ComponentModel2 InterfaceFault | InterfaceFault interfaceFaultComps @ NestedBaseValid elementDeclaration elementDeclIds

- Every Interface Fault component satisfies the base validity constraints.
- The Element Declaration component of each Interface Fault component is contained in the component model.

For each Interface Fault component in the interface faults property of an Interface component, the name property must be unique. Note that this constraint is enforced by the normative WSDL 2.0 XML schema.

Interface Fault components are uniquely identified by the QName of the enclosing Interface component and QName of the Interface Fault component itself.

Let InterfaceFaultKey express the QName uniqueness constraint on the Interface Fault component:

ComponentModel2 x, y : interfaceFaultComps | x.parent = y.parent x.name = y.name @ x = y

- No two Interface Fault components contained by the same Interface component have the same name property.

Despite having a name property, Interface Fault components cannot be identified solely by their QName. Indeed, two Interface components whose name property value has the same namespace name, but different local names, can contain Interface Fault components with the same name property value. Thus, the name property of Interface Fault component is not sufficient to form the unique identity of an Interface Fault component. A method for uniquely identifying components is defined in Appendix A.2 – Fragment Identifiers on page 84. See Appendix A.2.5 – The Interface Fault Component on page 88 for the definition of the fragment identifier for the Interface Fault component.

In cases where, due to an interface extending one or more other interfaces, two or more Interface Fault components have the same value for their name property, then the component models of those Interface Fault components MUST be equivalent (see § 2.15 – Equivalence of Components on page 62). If the Interface Fault components are equivalent then they are considered to collapse into a single component. Within the same Interface component, if two Interface Fault components are not equivalent then their name properties MUST NOT be equal.

Let InterfaceFaultNameUnique express the uniqueness constraint on the name property of an Interface Fault component among all the Interface Fault components available in an Interface component:

ComponentModel2 i : interfaceComps; x, y : interfaceFaultComps | x.id i.allInterfaceFaults y.id i.allInterfaceFaults x.name = y.name @ x = y

- No two Interface Fault components among all those available in the same Interface component have the same name property.

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

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

If a type system NOT based on the XML Infoset [XML Information Set] is in use (as considered in § 3.2 – Using Other Schema Languages on page 67) then additional properties would need to be added to the Interface Fault component (along with extension attributes to its XML representation) to allow associating such message types with the message reference.

Let InterfaceFaultCM be the conjunction of all the component model constraints on Interface Fault components.

InterfaceFaultCM InterfaceFaultRI InterfaceFaultKey InterfaceFaultNameUnique

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 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] on page 24.
  - An OPTIONAL element *attribute information item* as described below in § 2.3.2.2 – element *attribute information item* with fault [owner element] on page 24.
  - 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 on page 73).
  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 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 on page 24) to the properties of the Interface Fault component is as described in Mapping from XML Representation to Interface Fault Component Properties Property Value name 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 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 If the element attribute information item is present and its value is a QName, then the Element Declaration component from the element declarations property of the Description component resolved to by the value of the element attribute information item (see); otherwise empty. If the element attribute information item has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component. parent The Interface component corresponding to the interface element information item in [parent]. .

Mapping from XML Representation to Interface Fault Component Properties

Property	Value
name	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	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	If the element *attribute information item* is present and its value is a QName, then the Element Declaration component from the element declarations property of the Description component resolved to by the value of the element *attribute information item* (see § 2.17 – QName resolution on page 63); otherwise empty. If the element *attribute information item* has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component.
parent	The Interface component corresponding to the interface *element information item* in [parent].

2.4. Interface Operation

2.4.1. The Interface Operation Component

An Interface Operation component describes an operation that a given interface supports. An operation is an interaction with the service consisting of a set 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 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 and interface fault references properties). The service whose operation is using the pattern becomes one of the participants

of the pattern. This specification does not define a machine understandable language for defining message exchange patterns, nor does it define any specific patterns. The companion specification, [WSDL 2.0 Adjuncts] defines a set of such patterns and defines identifying IRIs any of which MAY be used as the value of the message exchange pattern property.

The properties of the Interface Operation component are as follows:

- name REQUIRED. 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]).
- interface message references OPTIONAL. A set of Interface Message Reference components for the ordinary messages the operation accepts or sends.
- interface fault references OPTIONAL. A set of Interface Fault Reference 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 properties of interface message references. (See § 2.4.1.2 – Operation Style on page 29.) These *xs:anyURI*s MUST be absolute IRIs (see [IETF RFC 3986]).
- parent REQUIRED. The Interface component that contains this component in its interface operations property.

Let InterfaceOperation be the set of all Interface Operation components:

NestedBase name : QName messageExchangePattern : AbsoluteURI interfaceMessageReferences : ID interfaceFaultReferences : ID style : AbsoluteURI
Each component referenced by an Interface Operation component must exist in the component model.

Let InterfaceOperationRI express the referential integrity constraints on the Interface Operation component:

ComponentModel2 InterfaceOperation | InterfaceOperation interfaceOpComps @ NestedBaseValid interfaceMessageReferences interfaceMessageRefIds interfaceFaultReferences interfaceFaultRefIds

- Every Interface Operation component satisfies the base validity constraints.
- The Interface Message Reference components of each Interface Operation component are contained in the component model.
- The Interface Fault Reference components of each Interface Operation component are contained in the component model.

For each Interface Operation component in the interface operations property of an Interface component, the name property MUST be unique. Note that this constraint is enforced by the normative WSDL 2.0 XML schema.

Interface Operation components are uniquely identified by the QName of the enclosing Interface component and QName of the Interface Operation component itself.

Let InterfaceOperationKey express the QName uniqueness constraint on the Interface Operation component:

ComponentModel2 x, y : interfaceOpComps | x.parent = y.parent x.name = y.name @ x = y

- No two Interface Operation components contained by the same Interface component have the same name property.

Despite having a name property, Interface Operation components cannot be identified solely by their QName. Indeed, two Interface components whose name property value has the same namespace name, but different local names, can contain Interface Operation components with the same name property value. Thus, the name property of Interface Operation components is not sufficient to form the unique identity of an Interface Operation component. A method for uniquely identifying components is defined in Appendix A.2 – Fragment Identifiers on page 84 . See Appendix A.2.6 – The Interface Operation Component on page 89 for the definition of the fragment identifier for the Interface Operation component.

In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation components have the same value for their name property, then the component models of those Interface Operation components MUST be equivalent (see § 2.15 – Equivalence of Components on page 62). If the Interface Operation components are equivalent then they are considered to collapse into a single component. Within the same Interface component, if two Interface Operation components are not equivalent then their name properties MUST NOT be equal.

Let InterfaceOperationNameUnique express the uniqueness constraint on the name property of an Interface Operation component among all the Interface Operation components available in an Interface component:

ComponentModel2 i : interfaceComps; x, y : interfaceOpComps | x.id i.allInterfaceOperations y.id i.allInterfaceOperations x.name = y.name @ x = y

- No two Interface Operation components among all those available in the same Interface component have the same name property.

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

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

More than one Interface Fault Reference component in the interface fault references property of an Interface Operation 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.

An Interface Operation component contains nested Interface Message Reference and Interface Fault Reference components. These components MUST have the Interface Operation component as their parent.

Let InterfaceOperationParent express the constraints on the parent properties of the nested components of an Interface Operation component:

ComponentModel2 io : interfaceOpComps; ifr : interfaceFaultRefComps; imr : interfaceMessageRefComps @ ifr.id io.interfaceFaultReferences ifr.parent = io.id imr.id io.interfaceMessageReferences imr.parent = io.id

- The set of Interface Fault Reference components contained by an Interface Operation component is exactly the set of Interface Fault Reference components that have that Interface Operation component as their parent.
- The set of Interface Message Reference components contained by an Interface Operation component is exactly the set of Interface Message Reference components that have that Interface Operation component as their parent.

Let InterfaceOperationCM be the conjunction of all the component model constraints on Interface Operation components.

InterfaceOperationCM InterfaceOperationRI InterfaceOperationKey InterfaceOperationParent Interface-OperationNameUnique

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] 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 component. Although a placeholder message is not itself a component, it is useful to regard it as having both a message label and a direction property which define the values of the actual Interface Message Reference 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.

Let Node be the set of all nodes that participate in message exchanges:

[Node]
Let PlaceholderMessage be the set of all placeholder messages:

messageLabel : NCName direction : Direction node : Node optional : Boolean
A *fault propagation ruleset* specifies the relation between the Interface Fault Reference and Interface Message Reference components of an Interface Operation component. The *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts* specification [WSDL 2.0 Adjuncts] 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]. Other message exchange patterns can define additional fault propagation rulesets.

Let FaultPropagationRuleset be the set of all fault propagation rulesets:

[FaultPropagationRuleset]
Let the predefined fault propagation rulesets be as follows:

messageTriggersFault : FaultPropagationRuleset faultReplacesMessage : FaultPropagationRuleset noFaults : FaultPropagationRuleset messageTriggersFault faultReplacesMessage faultReplacesMessage noFaults noFaults messageTriggersFault
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 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 property. A message exchange pattern is itself uniquely identified by an absolute IRI, which is used as the value of the message exchange pattern property of the Interface Operation component, and which specifies the fault propagation ruleset that its faults obey.

Let MessageExchangePattern be the set of all message exchange patterns:

messageExchangePattern : AbsoluteURI placeholderMessages : PlaceholderMessage faultPropagation-Ruleset : FaultPropagationRuleset placeholderMessages i1, i2 : ; p1, p2 : PlaceholderMessage | i1 p1 placeholderMessages i2 p2 placeholderMessages @ p1.messageLabel = p2.messageLabel i1 = i2

- Each message exchange pattern has at least one placeholder message.
- Each placeholder message in a message exchange pattern is uniquely identified by its message label.

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 property of the Interface Operation component contains a set of zero or more IRIs that identify operation styles. An Interface Operation component MUST satisfy the specification defined by each operation style identified by its style property. If no Interface Operation component can simultaneously satisfy all of the styles, the document is invalid.

If the style property of an Interface Operation 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 on page 67) of the Interface Message Reference or Interface Fault 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] 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 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] on page 30.
  - An OPTIONAL pattern *attribute information item* as described below in § 2.4.2.2 – pattern *attribute information item* with operation [owner element] on page 30.

- An OPTIONAL style *attribute information item* as described below in § 2.4.2.3 – style *attribute information item* with operation [owner element] on page 30.
- Zero or more namespace qualified *attribute information item*s 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 item*s (see § 5 – Documentation on page 73).
  2. One or more *element information item*s from among the following, in any order:
     - One or more *element information item*s from among the following, in any order:
       - Zero or more input *element information item*s (see § 2.5.2 – XML Representation of Interface Message Reference Component on page 33).
       - Zero or more output *element information item*s (see § 2.5.2 – XML Representation of Interface Message Reference Component on page 33).
       - Zero or more infault *element information item*s (see § 2.6.2 – XML Representation of Interface Fault Reference on page 37).
       - Zero or more outfault *element information item*s (see § 2.6.2 – XML Representation of Interface Fault Reference on page 37).
     - Zero or more namespace-qualified *element information item*s 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]).

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 properties of the Interface Message Reference components which are members of the interface message references 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]).

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 on page 29) to the properties of the Interface Operation component (see § 2.4.1 – The Interface Operation Component on page 25) is as described in Mapping from XML Representation to Interface Operation Component Properties Property Value name 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 The actual value of the pattern attribute information item; otherwise 'http://www.w3.org/ns/wsdl/in-out'. interface message references The set of message references corresponding to the input and output element information items in [children], if any. interface fault references The set of interface fault references corresponding to the infault and outfault element information items in [children], if any. style 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 The Interface component corresponding to the interface element information item in [parent]. .

Mapping from XML Representation to Interface Operation Component Properties

Property	Value
name	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	The actual value of the pattern *attribute information item*; otherwise 'http://www.w3.org/ns/wsdl/in-out'.
interface message references	The set of message references corresponding to the input and output *element information item*s in [children], if any.
interface fault references	The set of interface fault references corresponding to the infault and outfault *element information item*s in [children], if any.
style	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	The Interface 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 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 component is to associate an actual message element (XML element declaration or some other

declaration (see § 3.2 – Using Other Schema Languages on page 67)) 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 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 of the Interface Operation 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.
- 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. The direction MUST be the same as the direction of the message identified by the message label property in the message exchange pattern of the Interface Operation component this is contained within.
- message content model REQUIRED. An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*. A value of *#any* indicates that the message content is any single element. A value of *#none* indicates there is no message content. A value of *#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 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 component in the element declarations property of the Description component. This element represents the content or "payload" of the message. When the message content model property has the value *#any* or *#none*, the element declaration property MUST be empty.
- parent REQUIRED. The Interface Operation component that contains this component in its interface message references property.

Let Direction be a message direction of either *in* or *out*:

Direction ::= inToken | outToken
Let MessageContentModel be a message content model of either *any*, *none*, *other*, or *element*:

MessageContentModel ::= anyToken | noneToken | otherToken | elementToken
Let InterfaceMessageReference be the set of all Interface Message Reference components:

NestedBase messageLabel : NCName direction : Direction messageContentModel : MessageContentModel elementDeclaration : OPTIONAL[ID] messageContentModel = elementToken elementDeclaration

- The message content model is *element* exactly when the element declaration property is defined.

Each component referenced by an Interface Message Reference component must exist in the component model.

Let InterfaceMessageReferenceRI express the referential integrity constraints on the Interface Message Reference component:

ComponentModel2 InterfaceMessageReference | InterfaceMessageReference interfaceMessageRefComps @ NestedBaseValid elementDeclaration elementDeclIds

- Every Interface Message Reference component satisfies the base validity constraints.

- The Element Declaration components of each Interface Message Reference component are contained in the component model.

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

Let InterfaceMessageReferenceKey express this uniqueness constraint on the Interface Message Reference component:

ComponentModel2 x, y : interfaceMessageRefComps | x.parent = y.parent x.messageLabel = y.message-Label @ x = y

- No two Interface Message Reference components contained by the same Interface Operation component have the same message label property.

If a type system not based upon the XML Infoset is in use (as considered in § 3.2 – Using Other Schema Languages on page 67), then additional properties would need to be added to the Interface Message Reference component (along with extension attributes to its XML representation) to allow associating such message types with the message reference.

Let InterfaceMessageReferenceCM be the conjunction of all the component model constraints on Interface Message Reference components.

InterfaceMessageReferenceCM InterfaceMessageReferenceRI InterfaceMessageReferenceKey

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 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 item*s amongst its [attributes] as follows:
  - An OPTIONAL messageLabel *attribute information item* as described below in § 2.5.2.1 – messageLabel *attribute information item* with input or output [owner element] on page 34.
  - An OPTIONAL element *attribute information item* as described below in § 2.5.2.2 – element *attribute information item* with input or output [owner element] on page 34.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s 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 on page 33) to the properties of the Interface Message Reference component (see § 2.5.1 – The Interface Message Reference Component on page 31) is as described in Mapping from XML Representation to Interface Message Reference Component Properties Property Value message label The effective message label. direction The message direction. message content model If the element attribute information item is present and its value is a QName, then #element; otherwise the actual value of the element attribute information item, if any; otherwise #other. element declaration If the element attribute information item is present and its value is a QName, then the Element Declaration component from the element declarations property of the Description component resolved to by the value of the element attribute information item (see); otherwise empty. If the element attribute information item has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component. parent The Interface Operation component corresponding to the interface element information item in [parent]. and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the message exchange pattern of the parent Interface Operation 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 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 of some placeholder message with direction equal to the message direction.

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 equal to the message direction.

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 of the unique placeholder message with direction 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.

If the local name is output then the message exchange pattern MUST have at least one placeholder message with direction Out.

If the local name is infault then the message exchange pattern MUST support at least one fault in the In direction.

If the local name is outfault then the message exchange pattern MUST support at least one fault in the Out direction.

Mapping from XML Representation to Interface Message Reference Component Properties

Property	Value
message label	The effective message label.
direction	The message direction.
message content model	If the element *attribute information item* is present and its value is a QName, then *#element*; otherwise the actual value of the element *attribute information item*, if any; otherwise *#other*.
element declaration	If the element *attribute information item* is present and its value is a QName, then the Element Declaration component from the element declarations property of the Description component resolved to by the value of the element *attribute information item* (see § 2.17 – QName resolution on page 63); otherwise empty. If the element *attribute information item* has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component.
parent	The Interface Operation 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 component associates a defined type, specified by an Interface Fault component, to a fault message exchanged in an operation.

A message exchange pattern defines a set of placeholder messages that participate in the pattern and assigns them unique message labels within the pattern (e.g. 'In', 'Out'). The purpose of an Interface Fault Reference component is to associate an actual message type (XML element declaration or some other declaration (see § 3.2 – Using Other Schema Languages on page 67) for message content, as specified by an Interface Fault component) with a fault message occurring in the pattern. In order to identify the fault message it describes, the Interface Fault Reference 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] 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 component in the interface faults property of the [parent] Interface Operation component's [parent] Interface component, or an Interface component that it directly or indirectly extends. Identifying the Interface Fault 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 property of the Interface Operation component it is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
- direction REQUIRED. A *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. For example, if the ruleset *fault-replaces-message* is used, then a fault that refers to an outgoing message would have a direction 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 property value of *in* as the fault travels in the opposite direction of the message.
- parent REQUIRED. The Interface Operation component that contains this component in its interface fault references property.

Let InterfaceFaultReference be the set of all Interface Fault Reference components:

NestedBase interfaceFault : ID messageLabel : NCName direction : Direction
Each component referenced by a Interface Fault Reference component must exist in the component model.

Let InterfaceFaultReferenceRI express the referential integrity constraints on the Interface Fault Reference component:

ComponentModel2 InterfaceFaultReference | InterfaceFaultReference interfaceFaultRefComps @ NestedBaseValid interfaceFault interfaceFaultIds

- Every Interface Fault Reference component satisfies the base validity constraints.
- The Interface Fault component of each Interface Fault Reference component is contained in the component model.

For each Interface Fault Reference component in the interface fault references property of an Interface Operation component, the combination of its interface fault and message label properties MUST be unique.

Let InterfaceFaultReferenceKey express this uniqueness constraint on the Interface Fault Reference component:

ComponentModel2 x, y : interfaceFaultRefComps | x.parent = y.parent x.interfaceFault = y.interfaceFault x.messageLabel = y.messageLabel @ x = y

- No two Interface Fault Reference components contained by the same Interface Operation component have the same interface fault and message label properties.

An Interface Fault Reference component MUST refer to an Interface Fault component that is available in the associated Interface component. An Interface Fault component is available if it is contained in the Interface component or it is available in an Interface component that this Interface component extends.

Let InterfaceFaultReferenceConsistent express this consistency constraint on the Interface Fault Reference component:

ComponentModel2 ifr: interfaceFaultRefComps; io : interfaceOpComps; i : interfaceComps | ifr.parent = io.id io.parent = i.id @ ifr.interfaceFault i.allInterfaceFaults

- Every Interface Fault Reference component MUST refer to an Interface Fault component that is available in the Interface component that contains the Interface Operation component that contains the Interface Fault Reference component.

Let InterfaceFaultReferenceCM be the conjunction of all the component model constraints on Interface Fault Reference components.

InterfaceFaultReferenceCM InterfaceFaultReferenceRI InterfaceFaultReferenceKey InterfaceFaultReferenceConsistent

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 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 item*s amongst its [attributes] as follows:
  - A REQUIRED ref *attribute information item* as described below in § 2.6.2.1 – ref *attribute information item* with infault, or outfault [owner element] on page 38.
  - An OPTIONAL messageLabel *attribute information item* as described below in § 2.6.2.2 – messageLabel *attribute information item* with infault, or outfault [owner element] on page 38.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s 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 component with a given direction, if the message exchange pattern of the parent Interface Operation component has more than one fault with that direction. Recall that the *fault propagation ruleset* of the message exchange pattern 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 on page 37) to the properties of the Interface Fault Reference component (see § 2.6.1 – The Interface Fault Reference Component on page 36) is as described in Mapping from XML Representation to Interface Fault Reference Component Properties Property Value interface fault The Interface Fault component from interface faults property of the parent Interface component, or an Interface component that it directly or indirectly extends, with name equal to the actual value of the ref attribute information item. message label The effective message label. direction The fault direction. parent The Interface Operation component corresponding to the interface element information item in [parent]. and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the message exchange pattern of the parent Interface Operation 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 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 equal to the message direction.

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 of some placeholder message with direction equal to the message direction.

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 equal to the message direction.

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 of the unique placeholder message whose direction is equal to the message direction if the *attribute information item* is absent.

Mapping from XML Representation to Interface Fault Reference Component Properties

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

2.7. Binding

2.7.1. The Binding Component

A Binding component describes a concrete message format and transmission protocol which may be used to define an endpoint (see § 2.13 – Endpoint on page 58). That is, a Binding component defines the implementation details necessary to access the service.

Binding 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 on page 46) within an interface, in addition to across all operations of an interface.

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

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

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

A Binding component that defines bindings for an Interface component MUST define bindings for all the operations of that Interface component. 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]) or by defining bindings for each Interface Operation and Interface Fault component of the Interface component.

Similarly, whenever a reusable Binding component (i.e. one that does not specify an Interface component) is applied to a specific Interface component in the context of an Endpoint component (see § 2.13.1 – The Endpoint Component on page 58), the Binding component MUST define bindings for each Interface Operation and Interface Fault component of the Interface component, via a combination of properties defined on the Binding component itself and default binding rules specific to its binding type.

A Binding component that defines bindings for an Interface component MUST define bindings for all the faults of that Interface component that are referenced from any of the operations in that Interface component. 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 on page 63). For instance, Endpoint 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 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]. The value of this IRI indicates what kind of concrete binding details are contained within this Binding component. Specifications (such as [WSDL 2.0 Adjuncts]) 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 components.
- binding operations OPTIONAL. A set of Binding Operation components.

Let Binding be the set of all Binding components:

Base name : QName interface : OPTIONAL[ID] type : AbsoluteURI bindingFaults : ID bindingOperations : ID interface = bindingFaults = bindingOperations =

- If no Interface component is specified then there MUST NOT be any faults or operations defined.

Each component referenced by a Binding component must exist in the component model.

Let BindingRI express the referential integrity constraints on the Binding component:

ComponentModel2 Binding | Binding bindingComps @ BaseValid interface interfaceIds bindingFaults bindingFaultIds bindingOperations bindingOpIds

- Every Binding component satisfies the base validity constraints.
- The Interface component of each Binding component is contained in the component model.
- The Binding Fault components of each Binding component are contained in the component model.
- The Binding Operation components of each Binding component are contained in the component model.

For each Binding component in the bindings property of a Description component, the name property MUST be unique.

Let BindingKey express the QName uniqueness constraint on the Binding component:

ComponentModel2 x, y : bindingComps | x.name = y.name @ x = y

- No two Binding components have the same QName.

A Binding component contains nested Binding Operation and Binding Fault components. These components MUST have the Binding component as their parent.

Let BindingParent express the constraints on the parent properties of the nested components of a Binding component:

ComponentModel2 b : bindingComps; bf : bindingFaultComps; bo : bindingOpComps @ bf.id b.bindingFaults bf.parent = b.id bo.id b.bindingOperations bo.parent = b.id

- The set of Binding Fault components contained by a Binding component is exactly the set of Binding Fault components that have that Binding component as their parent.
- The set of Binding Operation components contained by a Binding component is exactly the set of Binding Operation components that have that Binding component as their parent.

Let BindingCM be the conjunction of all the component model constraints on Binding components.

BindingCM BindingRI BindingKey BindingParent

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 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 item*s 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] on page 42.
  - An OPTIONAL interface *attribute information item* as described below in § 2.7.2.2 – interface *attribute information item* with binding [owner element] on page 43.
  - An REQUIRED type *attribute information item* as described below in § 2.7.2.3 – type *attribute information item* with binding [owner element] on page 43.
  - 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], in order, as follows:
  1. Zero or more documentation *element information item*s (see § 5 – Documentation on page 73).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more fault *element information item*s (see § 2.8.2 – XML Representation of Binding Fault Component on page 45).
     - Zero or more operation *element information item*s (see § 2.9.2 – XML Representation of Binding Operation Component on page 48).
     - Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl. Such *element information item*s are considered to be binding extension elements(see § 2.7.2.4 – Binding extension elements on page 43).

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 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 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 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 on page 42) to the properties of the Binding component (see § 2.7.1 – The Binding Component on page 40) is as described in Mapping from XML Representation to Binding Component Properties Property Value name 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 The Interface component resolved to by the actual value of the interface attribute information item (see), if any. type The actual value of the type attribute information item. binding faults The set of Binding Fault components corresponding to the fault element information items in [children], if any. binding operations The set of Binding Operation components corresponding to the operation element information items in [children], if any. .

Mapping from XML Representation to Binding Component Properties

Property	Value
name	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	The Interface component resolved to by the actual value of the interface *attribute information item* (see § 2.17 – QName resolution on page 63), if any.
type	The actual value of the type *attribute information item*.

binding faults	The set of Binding Fault components corresponding to the fault *element information item*s in [children], if any.
binding operations	The set of Binding Operation components corresponding to the operation *element information item*s in [children], if any.

2.8. Binding Fault

2.8.1. The Binding Fault Component

A Binding Fault component describes a concrete binding of a particular fault within an interface to a particular concrete message format. A particular fault of an interface is uniquely identified by its name property.

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

The properties of the Binding Fault component are as follows:

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

Let BindingFault be the set of all Binding Fault components:

NestedBase interfaceFault : ID
Each component referenced by a Binding Fault component must exist in the component model.

Let BindingFaultRI express the referential integrity constraints on the Binding Fault component:

ComponentModel2 BindingFault | BindingFault bindingFaultComps @ NestedBaseValid interfaceFault interfaceFaultIds

- Every Binding Fault component satisfies the base validity constraints.
- The Interface Fault component of each Binding Fault component is contained in the component model.

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

Let BindingFaultKey express this uniqueness constraint on the Binding Fault component:

ComponentModel2 x, y : bindingFaultComps | x.parent = y.parent x.interfaceFault = y.interfaceFault @ x = y

- No two Binding Fault components contained by the same Binding component have the same interface fault property.

A Binding Fault component MUST bind an Interface Fault component that is available in the Interface component associated with the Binding component. An Interface Fault component is available if it is contained in the Interface component or is available in an extended Interface component.

Let BindingFaultConsistent express this consistency constraint on Binding Fault components:

ComponentModel2 bf : bindingFaultComps; b : bindingComps; i : interfaceComps | bf.parent = b.id b.interface = {i.id} @ bf.interfaceFault i.allInterfaceFaults

- Each Binding Fault component MUST bind an Interface Fault component that is available in the Interface component that is associated with its parent Binding component.

Let BindingFaultCM be the conjunction of all the component model constraints on Binding Fault components.

BindingFaultCM BindingFaultRI BindingFaultKey BindingFaultConsistent

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 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 ref *attribute information item* as described below in § 2.8.2.1 – ref *attribute information item* with fault [owner element] on page 45.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl. Such *element information item*s are considered to be binding fault extension elements as described further below (see § 2.8.2.2 – Binding Fault extension elements on page 46).

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 item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Fault component with additional properties and specify the mapping 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 on page 45) to the properties of the Binding Fault component (see § 2.8.1 – The Binding Fault Component on page 44) is as described in Mapping from XML Representation to Binding Fault Component Properties Property Value interface fault The Interface Fault component corresponding to the actual value of the ref attribute information item. parent The Binding component corresponding to the binding element information item in [parent]. .

Mapping from XML Representation to Binding Fault Component Properties

Property	Value
interface fault	The Interface Fault component corresponding to the actual value of the ref *attribute information item*.
parent	The Binding component corresponding to the binding *element information item* in [parent].

2.9. Binding Operation

2.9.1. The Binding Operation Component

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

The properties of the Binding Operation component are as follows:

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

Let BindingOperation be the set of all Binding Operation components:

NestedBase interfaceOperation : ID bindingMessageReferences : ID bindingFaultReferences : ID
Each component referenced by a Binding Operation component must exist in the component model.

Let BindingOperationRI express the referential integrity constraints on the Binding Operation component:

ComponentModel2 BindingOperation | BindingOperation bindingOpComps @ NestedBaseValid interfaceOperation interfaceOpIds bindingMessageReferences bindingMessageRefIds bindingFaultReferences bindingFaultRefIds

- Every Binding Operation component satisfies the base validity constraints.
- The Interface Operation component of each Binding Operation component is contained in the component model.
- The Binding Message Reference components of each Binding Operation component are contained in the component model.
- The Binding Fault Reference components of each Binding Operation component are contained in the component model.

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

Let BindingOperationKey express this uniqueness constraint on the Binding Operation component:

ComponentModel2 x, y : bindingOpComps | x.parent = y.parent x.interfaceOperation = y.interfaceOperation @ x = y

- No two Binding Operation components contained by the same Binding component have the same interface operation property.

A Binding Operation component contains nested Binding Message Reference and Binding Fault Reference components. These components MUST have the Binding Operation component as their parent.

Let BindingOperationParent express the constraints on the parent properties of the nested components of a Binding Operation component:

ComponentModel2 bo : bindingOpComps; bfr : bindingFaultRefComps; bmr : bindingMessageRefComps @ bfr.id bo.bindingFaultReferences bfr.parent = bo.id bmr.id bo.bindingMessageReferences bmr.parent = bo.id

- The set of Binding Fault Reference components contained by a Binding Operation component is exactly the set of Binding Fault Reference components that have that Binding Operation as their parent.
- The set of Binding Message Reference components contained by a Binding Operation component is exactly the set of Binding Message Reference components that have that Binding Operation as their parent.

A Binding Operation component MUST bind an Interface Operation component that is available in the Interface component associated with the Binding component. An Interface Operation component is available if it is contained in the Interface component or is available in an extended Interface component.

Let BindingOperationConsistent express this consistency constraint on Binding Operation components:

ComponentModel2 bo : bindingOpComps; b : bindingComps; i : interfaceComps | bo.parent = b.id b.interface = {i.id} @ bo.interfaceOperation i.allInterfaceOperations

- Each Binding Operation component MUST bind an Interface Operation component that is available in the Interface component that is associated with its parent Binding component.

Let BindingOperationCM be the conjunction of all the component model constraints on Binding Operation components.

BindingOperationCM BindingOperationRI BindingOperationKey BindingOperationParent BindingOperationConsistent

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 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 item*s 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] on page 48.
  - 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], in order, as follows:
  1. Zero or more documentation *element information item*s (see § 5 – Documentation on page 73).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more input *element information item*s (see § 2.10 – Binding Message Reference on page 49)
     - Zero or more output *element information item*s (see § 2.10 – Binding Message Reference on page 49)
     - Zero or more infault *element information item*s (see § 2.11 – Binding Fault Reference on page 52)
     - Zero or more outfault *element information item*s (see § 2.11 – Binding Fault Reference on page 52)
     - Zero or more namespace-qualified *element information item* whose [namespace name] is NOT http://www.w3.org/ns/wsdl . Such *element information item*s are considered to be binding operation extension elements as described below (see § 2.9.2.2 – Binding Operation extension elements on page 49).

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 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 on page 48) to the properties of the Binding Operation component is as described in Mapping from XML Representation to Binding Operation Component Properties Property Value interface operation The Interface Operation component corresponding to the actual value of the ref attribute information item. binding message references The set of Binding Message Reference components corresponding to the input and output element information items in [children], if any. binding fault references The set of Binding Fault Reference components corresponding to the infault and outfault element information items in [children], if any. parent The Binding component corresponding to the binding element information item in [parent]. .

Mapping from XML Representation to Binding Operation Component Properties

Property	Value
interface operation	The Interface Operation component corresponding to the actual value of the ref *attribute information item*.
binding message references	The set of Binding Message Reference components corresponding to the input and output *element information item*s in [children], if any.
binding fault references	The set of Binding Fault Reference components corresponding to the infault and outfault *element information item*s in [children], if any.
parent	The Binding 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 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 component among those in the interface message references property of the Interface Operation component being bound by the containing Binding Operation component.
- parent REQUIRED. The Binding Operation component that contains this component in its binding message references property.

Let BindingMessageReference be the set of all Binding Message Reference components:

NestedBase interfaceMessageReference : ID
Each component referenced by a Binding Message Reference component must exist in the component model.

Let BindingMessageReferenceRI express the referential integrity constraints on the Binding Message Reference component:

ComponentModel2 BindingMessageReference | BindingMessageReference bindingMessageRefComps @ NestedBaseValid interfaceMessageReference interfaceMessageRefIds

- Every Binding Message Reference component satisfies the base validity constraints.
- The Interface Message Reference component referred to by a Binding Message Reference component MUST exist in the component model.

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

Let BindingMessageReferenceKey express this uniqueness constraint on the Binding Message Reference component:

ComponentModel2 x, y : bindingMessageRefComps | x.parent = y.parent x.interfaceMessageReference = y.interfaceMessageReference @ x = y

- No two Binding Message Reference components contained by the same Binding Operation component have the same interface message reference property.

The Interface Message Reference component bound by a Binding Message Reference component MUST be contained in the Interface Operation component that is being bound by the Binding Operation that contains this Binding Message Reference component.

Let BindingMessageReferenceConsistent express this consistency constraint:

ComponentModel2 bmr : bindingMessageRefComps; bo : bindingOpComps; imr : interfaceMessageRef-Comps | bmr.parent = bo.id bmr.interfaceMessageReference = imr.id @ bo.interfaceOperation = imr.parent

- For each Binding Message Reference component, the parent Interface Operation component of its Interface Message Reference component is the Interface Operation component of its parent Binding Operation component.

Let BindingMessageReferenceCM be the conjunction of all the component model constraints on Binding Message Reference components.

BindingMessageReferenceCM BindingMessageReferenceRI BindingMessageReferenceKey BindingMes-sageReferenceConsistent

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 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 item*s 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] on page 51.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl. Such *element information item*s are considered to be binding message reference extension elements as described below (see § 2.10.2.2 – Binding Message Reference extension elements on page 51).

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 item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Message Reference component with additional properties and specify the mapping 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 on page 50) to the properties of the Binding Message Reference component is as described in Mapping from XML Representation to Binding Message Reference Component Properties Property Value interface message reference The Interface Message Reference component in the interface message references of the Interface Operation component being bound with message label equal to the effective message label. {parent} The Binding Operation component corresponding to the operation element information item in [parent]. and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the message exchange pattern of the Interface Operation 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 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 of some placeholder message with direction equal to the message direction.

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 equal to the message direction.

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 of the unique placeholder message with direction equal to the message direction if the *attribute information item* is absent.

Mapping from XML Representation to Binding Message Reference Component Properties

Property	Value
interface message reference	The Interface Message Reference component in the interface message references of the Interface Operation component being bound with message label equal to the effective message label.
{parent}	The Binding Operation 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 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 component among those in the interface fault references property of the Interface Operation component being bound by the parent Binding Operation component.
- parent REQUIRED. The Binding Operation component that contains this component in its binding fault references property.

Let BindingFaultReference be the set of all Binding Fault Reference components:

NestedBase interfaceFaultReference: ID
Each component referenced by a Binding Fault Reference component must exist in the component model.

Let BindingFaultReferenceRI express the referential integrity constraints on the Binding Fault Reference component:

ComponentModel2 BindingFaultReference | BindingFaultReference bindingFaultRefComps @ NestedBaseValid interfaceFaultReference interfaceFaultRefIds

- Every Binding Fault Reference component satisfies the base validity constraints.
- The Interface Fault Reference component referred to by a Binding Fault Reference component MUST exist in the component model.

For each Binding Fault Reference component in the binding fault references property of a Binding Operation component, the interface fault reference property MUST be unique. That is, the same fault cannot be bound twice within the same operation.

Let BindingFaultReferenceKey express this uniqueness constraint on the Binding Fault Reference component:

ComponentModel2 x, y : bindingFaultRefComps | x.parent = y.parent x.interfaceFaultReference = y.interfaceFaultReference @ x = y

- No two Binding Fault Reference components contained by the same Binding Operation component have the same interface fault reference property.

The Interface Fault Reference component bound by a Binding Fault Reference component MUST be contained in the Interface Operation component that is being bound by the Binding Operation that contains this Binding Fault Reference component.

Let BindingFaultReferenceConsistent express this consistency constraint:

ComponentModel2 bfr : bindingFaultRefComps; bo : bindingOpComps; ifr : interfaceFaultRefComps | bfr.parent = bo.id bfr.interfaceFaultReference = ifr.id @ bo.interfaceOperation = ifr.parent

- For each Binding Fault Reference component, the parent Interface Operation component of its Interface Fault Reference component is the Interface Operation component bound by its parent Binding Operation component.

Let BindingFaultReferenceCM be the conjunction of all the component model constraints on Binding Fault Reference components.

BindingFaultReferenceCM BindingFaultReferenceRI BindingFaultReferenceKey BindingFaultReferenceConsistent

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 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 item*s amongst its [attributes] as follows:
  - A REQUIRED ref *attribute information item* as described below in § 2.11.2.1 – ref *attribute information item* with infault or outfault [owner element] on page 54.

    An OPTIONAL messageLabel *attribute information item* as described below in § 2.11.2.2 – messageLabel *attribute information item* with infault or outfault [owner element] on page 54.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl. Such *element information item*s are considered to be binding fault reference extension elements as described below (see § 2.11.2.3 – Binding Fault Reference extension elements on page 54).

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 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 on page 53) to the properties of the Binding Fault Reference component is as described in Mapping from XML Representation to Binding Fault Reference Component Properties Property Value interface fault reference The Interface Fault Reference component in the interface fault references of the Interface Operation being bound with message label equal to the effective message label, and with interface fault equal to an Interface Fault component with name equal to the actual value of the ref attribute information item. {parent} The Binding Operation component corresponding to the operation element information item in [parent]. and uses the definitions below.

Define the *message exchange pattern* of the *element information item* to be the message exchange pattern of the Interface Operation 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 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 equal to the message direction.

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 of some placeholder message with direction equal to the message direction.

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 equal to the message direction.

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 of the unique placeholder message with direction equal to the message direction if the *attribute information item* is absent.

There MUST be an Interface Fault Reference component in the interface fault references of the Interface Operation being bound with message label equal to the effective message label and with interface fault equal to an Interface Fault component with name equal to the actual value of the ref *attribute information item*.

Mapping from XML Representation to Binding Fault Reference Component Properties

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

2.12. Service

2.12.1. The Service Component

A Service component describes a set of endpoints (see § 2.13 – Endpoint on page 58) 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 on page 63).

The properties of the Service component are as follows:

- name REQUIRED. An *xs:QName*.
- interface REQUIRED. An Interface component.
- endpoints REQUIRED. A non-empty set of Endpoint components.

Let Service be the set of all Service components:

Base name : QName interface : ID endpoints : ID
Each component referenced by a Service component must exist in the component model.

Let ServiceRI express the referential integrity constraints on the Service component:

ComponentModel2 Service | Service serviceComps @ BaseValid interface interfaceIds endpoints endpointIds

- Every Service component satisfies the base validity constraints.
- The Interface component of each Service component is contained in the component model.
- The Endpoint components of each Service component are contained in the component model.

For each Service component in the services property of a Description component, the name property MUST be unique.

Let ServiceKey express the QName uniqueness constraint on the Service component:

ComponentModel2 x, y : serviceComps | x.name = y.name @ x = y

- No two Service components have the same QName.

A Service component contains nested Endpoint components. These components MUST have the Service component as their parent.

Let ServiceParent express the constraints on the parent properties of the nested components of a Service component:

ComponentModel2 s : serviceComps; e : endpointComps @ e.id s.endpoints e.parent = s.id

- The set of Endpoint components contained by a Service component is exactly the set of Endpoint components that have that Service component as their parent.

Let ServiceCM be the conjunction of all the component model constraints on Service components.

ServiceCM ServiceRI ServiceKey ServiceParent

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 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 item*s 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] on page 57.
  - A REQUIRED interface *attribute information item* as described below in § 2.12.2.2 – interface *attribute information item* with service [owner element] on page 57.
  - Zero or more namespace qualified *attribute information item*s 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 item*s (see § 5 – Documentation on page 73).
  2. One or more *element information item*s from among the following, in any order:
     - One or more endpoint *element information item*s (see § 2.13.2 – XML Representation of Endpoint Component on page 59
     - Zero or more namespace-qualified *element information item*s 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 on page 57) to the properties of the Service component is as described in Mapping from XML Representation to Service Component Properties Property Value name 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 The Interface component resolved to by the actual value of the interface attribute information item (see). endpoints The Endpoint components corresponding to the endpoint element information items in [children]. .

Mapping from XML Representation to Service Component Properties

Property	Value
name	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	The Interface component resolved to by the actual value of the interface *attribute information item* (see § 2.17 – QName resolution on page 63).
endpoints	The Endpoint components corresponding to the endpoint *element information item*s in [children].

2.13. Endpoint

2.13.1. The Endpoint Component

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

Endpoint components are local to a given Service component (see Appendix A.2 – Fragment Identifiers on page 84).

The Binding component specified by the binding property of an Endpoint component is said to be applied to the Interface component which is the value of the interface property of the parent Service component of the Endpoint. According to the constraints given below, if this Binding component has an interface property, its value must be the Interface component the Binding component is applied to.

The address property is optional to allow for means other than IRIs to be used, e.g. a WS-Addressing Endpoint Reference [WSA 1.0 Core]. 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 component.
- address OPTIONAL. An *xs:anyURI*. This *xs:anyURI* MUST be an absolute IRI as defined by [IETF RFC 3987]. If present, the value of this attribute represents the network address at which the service

indicated by the parent Service component's interface property is offered via the binding referred to by the binding property. 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], section 6.8.2).

- parent REQUIRED. The Service component that contains this component in its endpoints property.

Let Endpoint be the set of all Endpoint components:

NestedBase name : NCName binding : ID address : OPTIONAL[AbsoluteURI]
Each component referenced by a Endpoint component must exist in the component model.

Let EndpointRI express the referential integrity constraints on the Endpoint component:

ComponentModel2 Endpoint | Endpoint endpointComps @ NestedBaseValid binding bindingIds

- Every Service component satisfies the base validity constraints.
- The Binding component of each Endpoint component is contained in the component model.

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

Let EndpointKey express this uniqueness constraint on the Endpoint component:

ComponentModel2 x, y : endpointComps | x.parent = y.parent x.name = y.name @ x = y

- No two Endpoint components contained by the same Service component have the same name property.

For each Endpoint component in the endpoints property of a Service component, the binding property MUST either be a Binding component with an unspecified interface property or a Binding component with an interface property equal to the interface property of the Service component.

Let EndpointConsistent express this consistency constraint:

ComponentModel2 s : serviceComps; e : endpointComps ; b : bindingComps | e.parent = s.id e.binding = b.id @ b.interface {s.interface}

- The Interface component of the Endpoint component's Service component MUST be the same as the Interface component of the Endpoint component's Binding component, if one is specified.

Let EndpointCM be the conjunction of all the component model constraints on Endpoint components.

EndpointCM EndpointRI EndpointKey EndpointConsistent

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 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 item*s amongst its [attributes] as follows:
  - A REQUIRED name *attribute information item* as described below in § 2.13.2.1 – name *attribute information item* with endpoint [owner element] on page 60.
  - A REQUIRED binding *attribute information item* as described below in § 2.13.2.2 – binding *attribute information item* with endpoint [owner element] on page 60.
  - An OPTIONAL address *attribute information item* as described below in § 2.13.2.3 – address *attribute information item* with endpoint [owner element] on page 60.
  - 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 on page 73).
  2. Zero or more namespace-qualified *element information item*s whose [namespace name] is NOT http://www.w3.org/ns/wsdl. Such *element information item*s are considered to be endpoint extension elements as described below (see § 2.13.2.4 – Endpoint extension elements on page 61).

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 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 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 on page 59) to the properties of the Endpoint component is as described in Mapping from XML Representation to Endpoint Component Properties Property Value name The actual value of the name attribute information item. binding The Binding component resolved to by the actual value of the binding attribute information item (see). address The actual value of the address attribute information item if present; otherwise empty. parent The Service component corresponding to the service element information item in [parent]. .

Mapping from XML Representation to Endpoint Component Properties

Property	Value
name	The actual value of the name *attribute information item*.
binding	The Binding component resolved to by the actual value of the binding *attribute information item* (see § 2.17 – QName resolution on page 63).
address	The actual value of the address *attribute information item* if present; otherwise empty.
parent	The Service 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] used in this specification are:

- *xs:token*
- *xs:NCName*
- *xs:anyURI*
- *xs:QName*
- *xs:boolean*

Let NCName be set of actual values of *xs:NCName*:

[NCName]
Let URI be the set of actual values of *xs:anyURI*:

[URI]
Let AbsoluteURI be the subset of absolute URIs (see [IETF RFC 3986]):

AbsoluteURI : URI

Let QName be the set of actual values of *xs:QName*:

- Let namespaceName be the namespace name.
- Let localName be the local name.

namespaceName : AbsoluteURI localName : NCName
Let Boolean be the set of actual values of *xs:boolean*:

Boolean ::= True | False

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 on page 61) 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], 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.

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

The Binding component specified by the binding property of an Endpoint is said to be applied to the Interface component which is the value of the interface property of the parent Service component for the Endpoint. Note that, if this Binding component has an interface property, then its value MUST be the Interface component that the Binding component is applied to.

2.16. Symbol Spaces

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

Within a symbol space, all qualified names (that is, the 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]. 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 property of the component.

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

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

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 on page 62), they MUST be compared character-by-character as indicated in [IETF RFC 3987].

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]. 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], [XML Schema: Datatypes].

Specifically, the element declarations and type definitions properties of the Description 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.

Support for the W3C XML Schema [XML Schema: Structures], [XML Schema: Datatypes] is included in the conformance criteria for WSDL 2.0 documents (see § 3.1 – Using W3C XML Schema Definition Language on page 64).

The schema components contained in the element declarations property of the Description component provide the type system used for Interface Message Reference and Interface Fault components. Interface Message Reference components indicate their structure and content by using the standard *attribute information item*s element, or for alternate schema languages in which these concepts do not map well, by using alternative *attribute information item* extensions. Interface Fault components behave similarly. Such extensions should define how they reference type system components. Such type system components MAY appear in additional collection properties on the Description 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 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 on page 73) 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]. 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 property of the Description 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.

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

Referenceability of schema components

	XML Representation	Referenceability of XML Schema Components
Including description	description/include	XML Schema components in the included Description component's element declarations and type definitions properties are referenceable.
Importing description	description/import	None of the XML Schema Components in the imported Description component are referenceable.
Importing XML Schema	description/types/xs:import	Element Declaration and Type Definition components in the imported namespace are referenceable.
Inlined XML Schema	description/types/xs:schema	Element Declaration and Type Definition 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], [XML Schema: Datatypes], 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 on page 63). 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 component or Type Definition component MUST appear in the element declarations or type definitions property respectively of the Description component corresponding to the WSDL document that imports the schema, or that imports directly or indirectly a WSDL document that imports the schema. Schema components not in an imported namespace MUST NOT appear in the element declarations or type definitions properties.

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

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

3.1.1.1. namespace *attribute information item*

The namespace *attribute information item* defines the namespace of the element declarations and type definitions imported from the referenced schema. The referenced schema MUST contain a targetNamespace *attribute information item* on its xs:schema *element information item*. 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. 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 on page 63). 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 on page 64).

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]. 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 on page 63). 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 component or Type Definition component MUST appear in the element declarations property or type definitions property respectively of the Description component corresponding to the WSDL document that contains the schema, or that imports directly or indirectly a WSDL document that contains the schema. Schema components not defined or declared in the inlined schema document or included by xs:include MUST NOT appear in the element declarations or type definitions properties.

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 on page 70 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] 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].

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. 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 or Interface Fault 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 on page 33) or fault *element information item* (see § 2.3.2 – XML Representation of Interface Fault Component on page 24). 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 and Interface Fault 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 and Interface Fault 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*). The extension specification SHOULD, if necessary, define additional properties of the Description component (and extension attributes) to hold the components of the referenced type system. It is expected that additional extension attributes for Interface Message Reference and Interface Fault 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. 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.

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

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 property of the Endpoint component. However, the use of these global *attribute information item*s 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]. See the primer [WSDL 2.0 Primer] 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 property of an Interface component.

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 property of a Binding component.

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 property of a Service component and the binding property of a nested Endpoint component, that is either the binding refers the interface of the service or the binding refers to no interface.

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

wsdlx:interface and wsdlx:binding are used to describe *element information item*s and *attribute information item*s whose type is xs:anyURI or a restriction of it, as well messages that contain the address property of an Endpoint. 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], 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 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] on page 70.
  - 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], as follows:
  - Zero or more documentation *element information item*s (see § 5 – Documentation on page 73).
  - Zero or more namespace-qualified *element information item*s 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. (see § 7 – Locating WSDL 2.0 Documents on page 76)

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*.

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], 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). 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 item*s.)

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*. 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 component to become broken references (see § 2.17 – QName resolution on page 63). 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 on page 63.

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 item*s amongst its [attributes] as follows:
  - A REQUIRED namespace *attribute information item* as described below in § 4.2.1 – namespace *attribute information item* on page 72.
  - An OPTIONAL location *attribute information item* as described below in § 4.2.2 – location *attribute information item* with import [owner element] on page 72.
  - 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 on page 73).
  - Zero or more namespace-qualified *element information item*s 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. If the location attribute in the import *element information item* is dereferencable, then it MUST reference a WSDL 2.0 document. 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 on page 76).

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>. 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]).

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 item*s ("mixed" content in XML Schema [XML Schema: Structures]). The documentation *element information item* is allowed inside any WSDL 2.0 *element information item*.

Like other *element information item*s in the http://www.w3.org/ns/wsdl namespace, the documentation *element information item* allows qualified *attribute information item*s whose [namespace name] is not http://www.w3.org/ns/wsdl. The xml:lang attribute (see [XML 1.0]) 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 item*s in its [attributes] property.
- Zero or more child *element information item*s in its [children] property.
- Zero or more *character information item*s in its [children] property.

6. Language Extensibility

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.

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. Base type for extension elements <xs:complexType name='ExtensionElement' abstract='true' > <xs:attribute ref='wsdl:required' use='optional' /> </xs:complexType> shows the type definition. The use of this type as a base type is optional.

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* on page 75) 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. 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.

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.

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.

☞ 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 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).

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 on page 1).

Authors of extension elements that may manifest as properties of the Description 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 on page 63). 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.

A normative XML Schema [XML Schema: Structures], [XML Schema: Datatypes] 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], 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]) for that namespace name. 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.

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 on page 1 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]. 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/.] Available at `http://www.w3.org/TR/2005/REC-charmod-20050215/`.

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.] 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.] 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. 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. 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/. Available at `http://www.w3.org/TR/2001/REC-xlink-20010627/`.

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. Available at `http://www.w3.org/TR/2006/REC-xml-20060816/`.

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. Available at `http://www.w3.org/TR/2006/REC-xml-names-20060816`.

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. Available at `http://www.w3.org/TR/2004/REC-xml-infoset-20040204`.

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. Available at `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

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. Available at `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`.

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. Available at `http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626`.

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. 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. 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. Available at `http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/`.

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. Available at `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

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. Available at `http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626`.

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. Available at `http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028`.

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. Available at `http://www.w3.org/TR/2005/NOTE-wsdl20-altschemalangs-20050817/`.

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/. Available at `http://www.w3.org/TR/2007/REC-soap12-part1-20070427/`.

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/. Available at `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`.

Z Notation Reference Manual

The Z Notation: A Reference Manual, Second Edition, J. M. Spivey, Prentice Hall, 1992. Available at `http://spivey.oriel.ox.ac.uk/mike/zrm/index.html`.

Fuzz 2000

Release Notes For Fuzz 2000, J. M. Spivey. Available at `http://spivey.oriel.ox.ac.uk/mike/fuzz/`.

Appendix 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].

Encoding considerations:

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

Security considerations:

See section Appendix A.3 – Security considerations on page 94.

Interoperability considerations:

There are no known interoperability issues.

Published specifications:

This document and [WSDL 2.0 Adjuncts].

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], section 5 or the syntax defined in Appendix A.2 – Fragment Identifiers on page 84.

Base URI:

As specified in [IETF RFC 3023], 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].

A WSDL 2.0 fragment identifier is an XPointer [XPointer], 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]). 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. 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 property of the component and the name properties of its ancestors as a path according to Rules for determining pointer parts for WSDL 2.0 components Component 1 2 3 4 Pointer Part Description n/a n/a n/a n/a wsdl.description() Element Declaration element QName n/a n/a n/a wsdl.elementDeclaration(element) Element Declaration element QName system IRI n/a n/a wsdl.elementDeclaration(element,system) Type Definition type QName n/a n/a n/a wsdl.typeDefinition(type) Type Definition type QName system IRI n/a n/a wsdl.typeDefinition(type,system) Interface interface NCName n/a n/a n/a wsdl.interface(interface) Interface Fault interface NCName fault NCName n/a n/a wsdl.interfaceFault(interface/fault) Interface Operation interface NCName operation NCName n/a n/a wsdl.interfaceOperation(interface/operation) Interface Message Reference interface NCName operation NCName message NCName n/a wsdl.interfaceMessageReference(interface/operation/message) Interface Fault Reference interface NCName operation NCName message NCName fault QName wsdl.interfaceFaultReference(interface/operation/message/fault) Binding binding NCName n/a n/a n/a wsdl.binding(binding) Binding Fault binding NCName fault QName n/a n/a wsdl.bindingFault(binding/fault) Binding Operation binding NCName operation QName n/a n/a wsdl.bindingOperation(binding/operation) Binding Message Reference binding NCName operation QName message NCName n/a wsdl.bindingMessageReference(binding/operation/message) Binding Fault Reference binding NCName operation QName message NCName fault QName wsdl.bindingFaultReference(binding/operation/message/fault) Service service NCName n/a n/a n/a wsdl.service(service) Endpoint service NCName endpoint NCName n/a n/a wsdl.endpoint(service/endpoint) Extensions namespace URI identifier extension-specific-syntax n/a n/a wsdl.extension(namespace,identifier) . 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 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 Rules for determining pointer parts for WSDL 2.0 components Component 1 2 3 4 Pointer Part Description n/a n/a n/a n/a wsdl.description() Element Declaration element QName n/a n/a n/a wsdl.elementDeclaration(element) Element Declaration element QName system IRI n/a n/a wsdl.elementDeclaration(element,system) Type Definition type QName n/a n/a n/a wsdl.typeDefinition(type) Type Definition type QName system IRI n/a n/a wsdl.typeDefinition(type,system) Interface interface NCName n/a n/a n/a wsdl.interface(interface) Interface Fault interface NCName fault NCName n/a n/a wsdl.interfaceFault(interface/fault) Interface Operation interface NCName operation NCName n/a n/a wsdl.interfaceOperation(interface/operation) Interface Message Reference interface NCName operation NCName message NCName n/a wsdl.interfaceMessageReference(interface/operation/message) Interface Fault Reference interface NCName operation NCName message NCName fault QName wsdl.interfaceFaultReference(interface/operation/message/fault) Binding binding NCName n/a n/a n/a wsdl.binding(binding) Binding Fault binding NCName fault QName n/a n/a wsdl.bindingFault(binding/fault) Binding Operation binding NCName operation QName n/a n/a wsdl.bindingOperation(binding/operation) Binding Message Reference binding NCName operation QName message NCName n/a wsdl.bindingMessageReference(binding/operation/message) Binding Fault Reference binding NCName operation QName message NCName fault QName wsdl.bindingFaultReference(binding/operation/message/fault) Service service NCName n/a n/a n/a wsdl.service(service) Endpoint service NCName endpoint NCName n/a n/a wsdl.endpoint(service/endpoint) Extensions namespace URI identifier extension-specific-syntax n/a n/a wsdl.extension(namespace,identifier) .

Rules for determining pointer parts for WSDL 2.0 components

Component	1	2	3	4	Pointer Part
Description	n/a	n/a	n/a	n/a	`wsdl.description()`
Element Declaration	*`element`* QName	n/a	n/a	n/a	`wsdl.elementDeclaration(`*`element`*`)`
Element Declaration	*`element`* QName	*`system`* IRI	n/a	n/a	`wsdl.elementDeclaration(`*`element,system`*`)`
Type Definition	*`type`* QName	n/a	n/a	n/a	`wsdl.typeDefinition(`*`type`*`)`
Type Definition	*`type`* QName	*`system`* IRI	n/a	n/a	`wsdl.typeDefinition(`*`type,system`*`)`
Interface	*`interface`* NCName	n/a	n/a	n/a	`wsdl.interface(`*`interface`*`)`
Interface Fault	*`interface`* NCName	*`fault`* NCName	n/a	n/a	`wsdl.interfaceFault(`*`interface/fault`*`)`
Interface Operation	*`interface`* NCName	*`operation`* NCName	n/a	n/a	`wsdl.interfaceOperation(`*`interface/operation`*`)`
Interface Message Reference	*`interface`* NCName	*`operation`* NCName	*`message`* NCName	n/a	`wsdl.interfaceMessageReference(`*`interface/operation/message`*`)`
Interface Fault Reference	*`interface`* NCName	*`operation`* NCName	*`message`* NCName	*`fault`* QName	`wsdl.interfaceFaultReference(`*`interface/operation/message/fault`*`)`
Binding	*`binding`* NCName	n/a	n/a	n/a	`wsdl.binding(`*`binding`*`)`
Binding Fault	*`binding`* NCName	*`fault`* QName	n/a	n/a	`wsdl.bindingFault(`*`binding/fault`*`)`
Binding Operation	*`binding`* NCName	*`operation`* QName	n/a	n/a	`wsdl.bindingOperation(`*`binding/operation`*`)`

Binding Message Reference	*binding* NCName	*operation* QName	*message* NCName	n/a	wsdl.bindingMessageReference(*binding/operation/message*)
Binding Fault Reference	*binding* NCName	*operation* QName	*message* NCName	*fault* QName	wsdl.bindingFaultReference(*binding/operation/message/fault*)
Service	*service* NCName	n/a	n/a	n/a	wsdl.service(*service*)
Endpoint	*service* NCName	*endpoint* NCName	n/a	n/a	wsdl.endpoint(*service/endpoint*)
Extensions	*namespace* URI	*identifier* extension-specific-syntax	n/a	n/a	wsdl.extension(*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.

Let ComponentID be the set of all component identifiers and component models that contain that identifier:

ComponentModel id : ID id componentIds
An IRI-reference consists of an IRI and a fragment identifier. IRI-references for WSDL 2.0 documents consist of an IRI that dereferences to a resource whose media type is `application/wsdl+xml` and a fragment identifier that conforms to XPointer syntax including the WSDL 2.0 pointer part schemes defined here. The interpretation of the WSDL 2.0 pointer parts is defined in terms of *component designators* which are themselves IRI-references. The component designator for a WSDL 2.0 document IRI-reference is formed by replacing the WSDL 2.0 document IRI by the target namespace IRI of the WSDL 2.0 document. The WSDL 2.0 pointer parts are interpreted in the context of the component model instance defined by the WSDL 2.0 document.

Let ComponentDesignator be the set of WSDL 2.0 component designators:

iri : AbsoluteURI fragId : wsdlPointerPart
We refer to the namespace of the WSDL 2.0 document as the *context namespace*. The Description, Element Declaration, and Type Definition components are not associated with any WSDL 2.0 namespace, however, for the purpose of constructing component designator IRI-references, we assign them the context namespace. In general, a WSDL 2.0 document may import other WSDL 2.0 namespaces, and the IRI of component designators for WSDL 2.0 components from the imported namespace is that namespace. Finally, the component model may contain extension components, in which case the specification for the extension must define the IRI used for extension component designators.

Let ComponentContext be the set of all component identifiers, component models that contain that identifier, and context namespaces:

ComponentID contextNamespace : AbsoluteURI
Let componentNamespace map a component within a given context to its component designator namespace IRI:

componentNamespace : ComponentContext AbsoluteURI
The namespace of a Description, Element Declaration, or Type Definition component is the context namespace:

ComponentContext | id descriptionIds elementDeclIds typeDefIds @ componentNamespace(ComponentContext) = contextNamespace
The namespace of an Interface, Binding, or Service component is the namespace of its name property:

ComponentContext; c : Component | c components id = Id(c) id interfaceIds bindingIds serviceIds @ componentNamespace(ComponentContext) = (Name(c)).namespaceName
The namespace of a nested component is equal to the namespace of its parent:

ComponentContext; c : NestedComponent | c components @ componentNamespace(ComponentContext) = componentNamespace(id : ID | id = ParentId(c) @ ComponentContext)
The syntax of extension component identifiers is defined by the corresponding extension specification.

Let ExtensionIdentifier be the set of all identifiers for extension components:

[ExtensionIdentifier]
Let wsdlPointerPart be the set of all WSDL 2.0 component pointer parts:

wsdlPointerPart ::= wsdlDescription | wsdlElementDeclarationQName OPTIONAL[AbsoluteURI] | wsdlTypeDefinitionQName OPTIONAL[AbsoluteURI] | wsdlInterfaceNCName | wsdlInterfaceFaultNCName NCName | wsdlInterfaceOperationNCName NCName | wsdlInterfaceMessageReferenceNCName NCName NCName | wsdlInterfaceFaultReferenceNCName NCName NCName QName | wsdlBindingNCName | wsdlBindingFaultNCName QName | wsdlBindingOperationNCName QName | wsdlBindingMessageReferenceNCName QName NCName | wsdlBindingFaultReferenceNCName QName NCName QName | wsdlServiceNCName | wsdlEndpointNCName NCName | wsdlExtensionAbsoluteURI ExtensionIdentifier
Let pointerPart map a component identifier within the context of some component model to its WSDL 2.0 pointer part:

pointerPart : ComponentID wsdlPointerPart
This map will be defined for each component in the following sections.

Let ComponentToDesignator map a WSDL 2.0 component to its component designator:

ComponentContext ComponentDesignator iri = componentNamespace(ComponentContext) fragId = pointerPart(ComponentID)

A.2.1. The Description Component

`wsdl.description()`

The description fragment identifier has no arguments and designates the unique Description component in the component model.

The pointer part defined by a Description component is wsdlDescription:

ComponentID | id descriptionIds @ pointerPart(ComponentID) = wsdlDescription

A.2.2. The Element Declaration Component

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

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

1. *`element`* is the name property of the Element Declaration component.
2. *`system`* is the namespace absolute IRI of the extension type system used for the Element Declaration component (see § 3.2 – Using Other Schema Languages on page 67). This parameter is absent if XML Schema is the type system.

Let xmlSchemaNamespaceURI be the namespace IRI for XML Schema:

xmlSchemaNamespaceURI : AbsoluteURI
Let ElementDeclArgs represent the arguments for the Element Declaration component pointer part:

element : QName system : OPTIONAL[AbsoluteURI] system {xmlSchemaNamespaceURI}
Let ElementDeclDesignator express the association between an Element Declaration component its pointer part arguments:

ComponentModel elementDeclComp : ElementDeclaration ElementDeclArgs elementDeclComp elementDeclComps elementDeclComp.name = element elementDeclComp.system {xmlSchemaNamespaceURI} system
The pointer part defined by an Element Declaration component is wsdlElementDeclaration:

ElementDeclDesignator; id : ID | id = elementDeclComp.id @ pointerPart(ComponentID) = wsdlElementDeclaration(element, system)

A.2.3. The Type Definition Component

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

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

1. *`type`* is the name property of the Type Definition component.
2. *`system`* is the namespace absolute IRI of the extension type system used for the Type Definition component (see § 3.2 – Using Other Schema Languages on page 67). This parameter is absent if XML Schema is the type system.

Let TypeDefArgs represent the arguments for the Type Definition component pointer part:

type : QName system : OPTIONAL[AbsoluteURI] system {xmlSchemaNamespaceURI}
Let TypeDefDesignator express the association between a Type Definition component and its pointer part arguments:

ComponentModel typeDefComp : TypeDefinition TypeDefArgs typeDefComp typeDefComps typeDefComp.name = type typeDefComp.system {xmlSchemaNamespaceURI} system
The pointer part defined by a Type Definition component is wsdlTypeDefinition:

TypeDefDesignator; id : ID | id = typeDefComp.id @ pointerPart(ComponentID) = wsdlTypeDefinition(type, system)

A.2.4. The Interface Component

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

1. *`interface`* is the local name of the name property of the Interface component.

Let InterfaceArgs represent the arguments for the Interface component pointer part:

interface : NCName
Let InterfaceDesignator express the association between an Interface component and its pointer part arguments:

ComponentModel interfaceComp : Interface InterfaceArgs interfaceComp interfaceComps interfaceComp.name.localName = interface
The pointer part defined by an Interface component is wsdlInterface:

InterfaceDesignator; id : ID | id = interfaceComp.id @ pointerPart(ComponentID) = wsdlInterface(interface)

A.2.5. The Interface Fault Component

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

1. *interface* is the local name of the name property of the parent Interface component.
2. *fault* is the local name of the name property of the Interface Fault component.

Let InterfaceFaultArgs represent the arguments for the Interface Fault component pointer part:

InterfaceArgs fault : NCName
Let InterfaceFaultDesignator express the association between an Interface Fault component and its pointer part arguments:

InterfaceDesignator interfaceFaultComp : InterfaceFault InterfaceFaultArgs interfaceFaultComp interfaceFaultComps interfaceFaultComp.parent = interfaceComp.id interfaceFaultComp.name.localName = fault
The pointer part defined by an Interface Fault component is wsdlInterfaceFault:

InterfaceFaultDesignator; id : ID | id = interfaceFaultComp.id @ pointerPart(ComponentID) = wsdlInterfaceFault(interface, fault)

A.2.6. The Interface Operation Component

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

1. *interface* is the local name of the name property of the parent Interface component.
2. *operation* is the local name of the name property of the Interface Operation component.

Let InterfaceOpArgs represent the arguments for the Interface Operation component pointer part:

InterfaceArgs operation : NCName
Let InterfaceOpDesignator express the association between an Interface Operation component and its pointer part arguments:

InterfaceDesignator interfaceOpComp : InterfaceOperation InterfaceOpArgs interfaceOpComp interfaceOpComps interfaceOpComp.parent = interfaceComp.id interfaceOpComp.name.localName = operation
The pointer part defined by an Interface Operation component is wsdlInterfaceOperation:

InterfaceOpDesignator; id : ID | id = interfaceOpComp.id @ pointerPart(ComponentID) = wsdlInterfaceOperation(interface, operation)

A.2.7. The Interface Message Reference Component

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

1. *interface* is the local name of the name property of the grandparent Interface component.
2. *operation* is the local name of the name property of the parent Interface Operation component.
3. *message* is the message label property of the Interface Message Reference component.

Let InterfaceMessageRefArgs represent the arguments for the Interface Message Reference pointer part:

InterfaceOpArgs message : NCName
Let InterfaceMessageRefDesignator express the association between an Interface Message Reference component and its pointer part arguments:

InterfaceOpDesignator interfaceMessageRefComp : InterfaceMessageReference InterfaceMessageRefArgs interfaceMessageRefComp interfaceMessageRefComps interfaceMessageRefComp.parent = interfaceOpComp.id interfaceMessageRefComp.messageLabel = message
The pointer part defined by an Interface Message Reference component is wsdlInterfaceMessageReference:

InterfaceMessageRefDesignator; id : ID | id = interfaceMessageRefComp.id @ pointerPart(ComponentID) = wsdlInterfaceMessageReference(interface, operation, message)

A.2.8. The Interface Fault Reference Component

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

1. *`interface`* is the local name of the name property of the grandparent Interface component.
2. *`operation`* is the local name of the name property of the parent Interface Operation component.
3. *`message`* is the message label property of the Interface Fault Reference component.
4. *`fault`* is the name property of the Interface Fault component referred to by the interface fault property of the Interface Fault Reference component.

Let InterfaceFaultRefArgs represent the arguments for the Interface Fault Reference component pointer part:

InterfaceOpArgs message : NCName fault : QName
Let InterfaceFaultRefDesignator express the association between an Interface Fault Reference component and its pointer part arguments:

InterfaceOpDesignator interfaceFaultComp : InterfaceFault interfaceFaultRefComp : InterfaceFaultReference InterfaceFaultRefArgs interfaceFaultComp interfaceFaultComps interfaceFaultComp.id interface-Comp.allInterfaceFaults interfaceFaultComp.name = fault interfaceFaultRefComp interfaceFaultRefComps interfaceFaultRefComp.interfaceFault = interfaceFaultComp.id interfaceFaultRefComp.messageLabel = message
The pointer part defined by an Interface Fault Reference component is wsdlInterfaceFaultReference:

InterfaceFaultRefDesignator; id : ID | id = interfaceFaultRefComp.id @ pointerPart(ComponentID) = wsdlInterfaceFaultReference(interface, operation, message, fault)

A.2.9. The Binding Component

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

1. *`binding`* is the local name of the name property of the Binding component.

Let BindingArgs represent the arguments for the Binding component pointer part:

binding : NCName
Let BindingDesignator express the association between a Binding component and its pointer part arguments:

ComponentModel bindingComp : Binding BindingArgs bindingComp bindingComps binding-Comp.name.localName = binding
Note that the above definition applies to all Binding components, whether or not they bind a specific Interface component.

The pointer part defined by a Binding component is wsdlBinding:

BindingDesignator; id : ID | id = bindingComp.id @ pointerPart(ComponentID) = wsdlBinding(binding)
Let BindingInterfaceDesignator express the association between an Interface component and the pointer part arguments in the case that the associated Binding component binds a specific Interface component.

BindingDesignator interfaceComp : Interface interfaceComp interfaceComps bindingComp.interface = {interfaceComp.id}

A.2.10. The Binding Fault Component

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

1. *`binding`* is the local name of the name property of the parent Binding component.
2. *`fault`* is the name property of the Interface Fault component referred to by the interface fault property of the Binding Fault component.

Let BindingFaultArgs represent the arguments for the Binding Fault pointer part:

BindingArgs fault : QName
Let BindingFaultDesignator express the association between a Binding Fault component and its pointer part arguments:

BindingInterfaceDesignator interfaceFaultComp : InterfaceFault bindingFaultComp : BindingFault BindingFaultArgs interfaceFaultComp interfaceFaultComps interfaceFaultComp.id interfaceComp.allInterfaceFaults interfaceFaultComp.name = fault bindingFaultComp bindingFaultComps bindingFaultComp.parent = bindingComp.id bindingFaultComp.interfaceFault = interfaceFaultComp.id
The pointer part defined by a Binding Fault component is wsdlBindingFault:

BindingFaultDesignator; id : ID | id = bindingFaultComp.id @ pointerPart(ComponentID) = wsdlBindingFault(binding, fault)

A.2.11. The Binding Operation Component

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

1. *`binding`* is the local name of the name property of the parent Binding component.
2. *`operation`* is the name property of the Interface Operation component referred to by the interface operation property of the Binding Operation component.

Let BindingOpArgs represent the arguments for the Binding Operation component:

BindingArgs operation : QName
Let BindingOpDesignator express the association between a Binding Operation component and its pointer part arguments:

BindingInterfaceDesignator interfaceOpComp : InterfaceOperation bindingOpComp : BindingOperation BindingOpArgs interfaceOpComp interfaceOpComps interfaceOpComp.id interfaceComp.allInterfaceOperations interfaceOpComp.name = operation bindingOpComp bindingOpComps bindingOpComp.parent = bindingComp.id bindingOpComp.interfaceOperation = interfaceOpComp.id
The pointer part defined by a Binding Operation component is wsdlBindingOperation:

BindingOpDesignator; id : ID | id = bindingOpComp.id @ pointerPart(ComponentID) = wsdlBindingOperation(binding, operation)

A.2.12. The Binding Message Reference Component

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

1. *`binding`* is the local name of the name property of the grandparent Binding component.
2. *`operation`* is the name property of the Interface Operation component referred to by the interface operation property of the parent Binding Operation component.

3. *`message`* is the message label property of the Interface Message Reference component referred to by the interface message reference property of the Binding Message Reference component.

Let BindingMessageRefArgs represent the arguments for the Binding Message Reference pointer part:

BindingOpArgs message : NCName
Let BindingMessageRefDesignator express the association between a Binding Message Reference component and its pointer part arguments:

BindingOpDesignator interfaceMessageRefComp : InterfaceMessageReference bindingMessageRefComp : BindingMessageReference BindingMessageRefArgs interfaceMessageRefComp interfaceMessageRefComps interfaceMessageRefComp.parent = interfaceOpComp.id interfaceMessageRefComp.messageLabel = message bindingMessageRefComp bindingMessageRefComps bindingMessageRefComp.parent = bindingOpComp.id bindingMessageRefComp.interfaceMessageReference = interfaceMessageRefComp.id
The pointer part defined by a Binding Message Reference component is wsdlBindingMessageReference:

BindingMessageRefDesignator; id : ID | id = bindingMessageRefComp.id @ pointerPart(ComponentID) = wsdlBindingMessageReference(binding, operation, message)

A.2.13. The Binding Fault Reference Component

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

1. *`binding`* is the local name of the name property of the grandparent Binding component.
2. *`operation`* is the name property of the Interface Operation component referred to by the interface operation property of the parent Binding Operation component.
3. *`message`* is the message label property of the Interface Fault Reference component referred to by the interface fault reference property of the Binding Fault Reference component.
4. *`fault`* is the name property of the Interface Fault component referred to by the interface fault property of the Interface Fault Reference component referred to by the interface fault reference property of the Binding Fault Reference component.

Let BindingFaultRefArgs represent the arguments for the Binding Fault Reference pointer part:

BindingOpArgs BindingFaultArgs message : NCName
Let BindingFaultRefDesignator express the association between a Binding Fault Reference component and its pointer part arguments:

BindingOpDesignator BindingFaultDesignator interfaceFaultRefComp : InterfaceFaultReference bindingFaultRefComp : BindingFaultReference BindingFaultRefArgs interfaceFaultRefComp interfaceFaultRefComps interfaceFaultRefComp.parent = interfaceOpComp.id interfaceFaultRefComp.interfaceFault = interfaceFaultComp.id interfaceFaultRefComp.messageLabel = message bindingFaultRefComp bindingFaultRefComps bindingFaultRefComp.parent = bindingOpComp.id bindingFaultRefComp.interfaceFaultReference = interfaceFaultRefComp.id
The pointer part defined by a Binding Fault Reference component is wsdlBindingFaultReference:

BindingFaultRefDesignator; id : ID | id = bindingFaultRefComp.id @ pointerPart(ComponentID) = wsdlBindingFaultReference(binding, operation, message, fault)

A.2.14. The Service Component

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

1. *`service`* is the local name of the name property of the Service component.

Let ServiceArgs represent the arguments for the Service pointer part:

service : NCName
Let ServiceDesignator express the association between a Service component and its pointer part arguments:

ComponentModel serviceComp : Service ServiceArgs serviceComp serviceComps serviceComp.name.local-Name = service
The pointer part defined by a Service component is wsdlService:

ServiceDesignator; id : ID | id = serviceComp.id @ pointerPart(ComponentID) = wsdlService(service)

A.2.15. The Endpoint Component

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

1. *`service`* is the local name of the name property of the parent Service component.
2. *`endpoint`* is the name property of the Endpoint component.

Let EndpointArgs represent the arguments for the Endpoint pointer part:

ServiceArgs endpoint : NCName
Let EndpointDesignator express the association between an Endpoint component and its pointer part arguments:

ServiceDesignator endpointComp : Endpoint EndpointArgs endpointComp endpointComps endpoint-Comp.parent = serviceComp.id endpointComp.name = endpoint
The pointer part defined by a Endpoint component is wsdlEndpoint:

EndpointDesignator; id : ID | id = endpointComp.id @ pointerPart(ComponentID) = wsdlEndpoint(service, endpoint)

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.

Let ExtensionArgs represent the arguments for the extension component pointer part:

namespace : AbsoluteURI identifier : ExtensionIdentifier
Let ExtensionDesignator express the association between an extension component its pointer part arguments:

ComponentModel extensionComp : Component ExtensionArgs extensionComp components
The details of the association are defined by each extension specification.

The namespace IRI of an extension component is defined by the extension specification.

The pointer part defined by an extension component is wsdlExtension:

ExtensionDesignator; id : ID | id = Id(extensionComp) @ pointerPart(ComponentID) = wsdlExtension(namespace, identifier)

A.3. Security considerations

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

Appendix 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.

Appendix 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 on page 10 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 property of either the component itself, in the case of Interface , Binding , and Service components, or the name property of the ancestor top-level component. The IRI provided by the namespace name of the name property is combined with a zero or more `xmlns` pointer parts (see *3.4 Namespace Binding Context* in [XPointer]) followed by a single WSDL 2.0 pointer part as defined in Appendix A.2 – Fragment Identifiers on page 84 .

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

C.3. Example

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

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:

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)
```

Appendix D. Component Summary (Non-Normative)

Summary of WSDL 2.0 Components and their Properties Component Defined Properties name, parent Binding binding faults, binding operations, interface, name, type Binding Fault interface fault, parent Binding Fault Reference interface fault reference, parent Binding Message Reference interface message reference, parent Binding Operation binding fault references, binding message references, interface operation, parent Description bindings, element declarations, interfaces, services, type definitions Element Declaration name, system Endpoint address, binding, name, parent Interface extended interfaces, interface faults, interface operations, name Interface Fault element declaration, message content model, name, parent Interface Fault Reference direction, interface fault, message label, parent Interface Message Reference direction, element declaration, message content model, message label, parent Interface Operation interface

fault references, interface message references, message exchange pattern, name, parent, style Service endpoints, interface, name Type Definition name, system Property Where Defined address Endpoint.address binding Endpoint.binding binding fault references Binding Operation. binding fault references binding faults Binding.binding faults binding message references Binding Operation. binding message references binding operations Binding.binding operations bindings Description.bindings direction Interface Fault Reference.direction, Interface Message Reference.direction element declaration Interface Fault.element declaration, Interface Message Reference.element declaration element declarations Description.element declarations endpoints Service.endpoints extended interfaces Interface.extended interfaces interface Binding.interface, Service.interface interface fault Binding Fault. interface fault , Interface Fault Reference.interface fault interface fault reference Binding Fault Reference. interface fault reference interface fault references Interface Operation.interface fault references interface faults Interface.interface faults interface message reference Binding Message Reference. interface message reference interface message references Interface Operation.interface message references interface operation Binding Operation. interface operation interface operations Interface.interface operations interfaces Description.interfaces message content model Interface Fault.message content model, Interface Message Reference.message content model message exchange pattern Interface Operation.message exchange pattern message label Interface Fault Reference.message label, Interface Message Reference.message label name .name, Binding.name, Element Declaration.name, Endpoint.name, Interface.name, Interface Fault.name, Interface Operation.name, Service.name, Type Definition.name parent .parent, Binding Fault.parent, Binding Fault Reference. parent , Binding Message Reference. parent , Binding Operation.parent, Endpoint.parent, Interface Fault.parent, Interface Fault Reference.parent, Interface Message Reference.parent, Interface Operation.parent services Description.services style Interface Operation.style system Element Declaration.system, Type Definition.system type Binding.type type definitions Description.type definitions 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.

Summary of WSDL 2.0 Components and their Properties

Component	Defined Properties
	name, parent
Binding	binding faults, binding operations, interface, name, type
Binding Fault	interface fault, parent
Binding Fault Reference	interface fault reference, parent
Binding Message Reference	interface message reference, parent
Binding Operation	binding fault references, binding message references, interface operation, parent
Description	bindings, element declarations, interfaces, services, type definitions
Element Declaration	name, system
Endpoint	address, binding, name, parent
Interface	extended interfaces, interface faults, interface operations, name
Interface Fault	element declaration, message content model, name, parent
Interface Fault Reference	direction, interface fault, message label, parent
Interface Message Reference	direction, element declaration, message content model, message label, parent
Interface Operation	interface fault references, interface message references, message exchange pattern, name, parent, style
Service	endpoints, interface, name

Type Definition	name, system
Property	**Where Defined**
address	Endpoint.address
binding	Endpoint.binding
binding fault references	Binding Operation. binding fault references
binding faults	Binding.binding faults
binding message references	Binding Operation. binding message references
binding operations	Binding.binding operations
bindings	Description.bindings
direction	Interface Fault Reference.direction, Interface Message Reference.direction
element declaration	Interface Fault.element declaration, Interface Message Reference.element declaration
element declarations	Description.element declarations
endpoints	Service.endpoints
extended interfaces	Interface.extended interfaces
interface	Binding.interface, Service.interface
interface fault	Binding Fault. interface fault , Interface Fault Reference.interface fault
interface fault reference	Binding Fault Reference. interface fault reference
interface fault references	Interface Operation.interface fault references
interface faults	Interface.interface faults
interface message reference	Binding Message Reference. interface message reference
interface message references	Interface Operation.interface message references
interface operation	Binding Operation. interface operation
interface operations	Interface.interface operations
interfaces	Description.interfaces
message content model	Interface Fault.message content model, Interface Message Reference.message content model
message exchange pattern	Interface Operation.message exchange pattern
message label	Interface Fault Reference.message label, Interface Message Reference.message label
name	.name, Binding.name, Element Declaration.name, Endpoint.name, Interface.name, Interface Fault.name, Interface Operation.name, Service.name, Type Definition.name
parent	.parent, Binding Fault.parent, Binding Fault Reference. parent , Binding Message Reference. parent , Binding Operation.parent, Endpoint.parent, Interface Fault.parent, Interface Fault Reference.parent, Interface Message Reference.parent, Interface Operation.parent
services	Description.services
style	Interface Operation.style
system	Element Declaration.system, Type Definition.system
type	Binding.type
type definitions	Description.type definitions

Appendix 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.

Summary of Assertions about WSDL 2.0 Documents

Id	Assertion
	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 on page 69), 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.
	Zero or more *element information item*s amongst its [children], in order as follows:
	Its value MUST be an absolute IRI (see [IETF RFC 3987]) and should be dereferencable.
	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).
	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*.
	This value MUST NOT match the actual value of targetNamespace *attribute information item* in the enclosing WSDL 2.0 document.
	If the location attribute in the import *element information item* is dereferencable, then it MUST reference a WSDL 2.0 document.
	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 on page 76).
	The IRI indicated by location MUST resolve to a WSDL 2.0 document.
	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*.
	Its value, if present, MUST contain absolute IRIs (see [IETF RFC 3987]).
	If the element *attribute information item* has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component.
	The messageLabel *attribute information item* MUST be present in the XML representation of an Interface Fault Reference component with a given direction, if the message exchange pattern of the parent Interface Operation component has more than one fault with that direction.
	If the element *attribute information item* has a value, then it MUST resolve to an Element Declaration component from the element declarations property of the Description component.
	It MUST NOT appear on a wsdl:description element or any of its children/descendants.
	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.
	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 of some placeholder message with direction equal to the message direction.
	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 equal to the message direction.

	If the local name is input then the message exchange pattern MUST have at least one placeholder message with direction In.
	If the local name is output then the message exchange pattern MUST have at least one placeholder message with direction Out.
	If the local name is infault then the message exchange pattern MUST support at least one fault in the In direction.
	If the local name is outfault then the message exchange pattern MUST support at least one fault in the Out direction.
	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 equal to the message direction.
	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 of some placeholder message with direction equal to the message direction.
	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 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 of some placeholder message with direction equal to the message direction.
	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 equal to the message direction.
	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 equal to the message direction.
	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 of some placeholder message with direction equal to the message direction.
	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 equal to the message direction.
	A Description component MUST NOT have such broken references.
	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].
	The referenced schema MUST contain a targetNamespace *attribute information item* on its xs:schema *element information item*.
	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.
	A WSDL 2.0 document MUST NOT define the same element or type in more than one inlined schema.
	A specification of extension syntax for an alternative schema language MUST use a namespace that is different than the namespace of XML Schema.
	The namespace used for an alternate schema language MUST be an absolute IRI.
	If wsdlx:interface and wsdlx:binding are used together then they MUST satisfy the same consistency rules that apply to the interface property of a Service component and the binding property of a nested Endpoint component, that is either the binding refers the interface of the service or the binding refers to no interface.

	A specification of extension syntax for an alternative schema language MUST include the declaration of an *element information item*, intended to appear as a child of the wsdl:types *element information item*, which references, names, and locates the schema instance (an import *element information item*).
	The type of the wsdlx:interface *attribute information item* is an *xs:QName* that specifies the name property of an Interface component.
	The type of the wsdlx:binding *attribute information item* is an *xs:QName* that specifies the name property of a Binding component.

Summary of Assertions about WSDL 2.0 Components

Id	Assertion
	If a Binding component specifies any operation-specific binding details (by including Binding Operation components) or any fault binding details (by including Binding Fault components), then it MUST specify an interface the Binding component applies to, so as to indicate which interface the operations come from.
	A Binding component that defines bindings for an Interface component MUST define bindings for all the operations of that Interface component.
	Similarly, whenever a reusable Binding component (i.e. one that does not specify an Interface component) is applied to a specific Interface component in the context of an Endpoint component (see § 2.13.1 – The Endpoint Component on page 58), the Binding component MUST define bindings for each Interface Operation and Interface Fault component of the Interface component, via a combination of properties defined on the Binding component itself and default binding rules specific to its binding type.
	A Binding component that defines bindings for an Interface component MUST define bindings for all the faults of that Interface component that are referenced from any of the operations in that Interface component.
	This *xs:anyURI* MUST be an absolute IRI as defined by [IETF RFC 3987].
	For each Binding component in the bindings property of a Description component, the name property MUST be unique.
	For each Binding Fault component in the binding faults property of a Binding component, the interface fault property MUST be unique.
	For each Binding Fault Reference component in the binding fault references property of a Binding Operation component, the interface fault reference property MUST be unique.
	There MUST be an Interface Fault Reference component in the interface fault references of the Interface Operation being bound with message label equal to the effective message label and with interface fault equal to an Interface Fault component with name equal to the actual value of the ref *attribute information item*.
	For each Binding Message Reference component in the binding message references property of a Binding Operation component, the interface message reference property MUST be unique.
	For each Binding Operation component in the binding operations property of a Binding component, the interface operation property MUST be unique.
	The fragment identifier consists of a sequence zero or more `xmlns()` pointer parts followed by exactly one `wsdl.*()` pointer part.
	Each `xmlns()` pointer part that appears in the fragment identifier defines a namespace that is referenced by the `wsdl.*()` pointer part.
	Each `xmlns()` pointer part defines a unique namespace.
	The `xmlns()` pointer parts define namespaces in the same order as they are referenced in the `wsdl.*()` pointer part.
	The namespace prefixes defined by the `xmlns()` pointer parts are named `ns1`, `ns2`, etc., in the order of their appearance.

	The fragment identifier contains no optional whitespace.
	No `xmlns()` pointer part defines a namespace for the targetNamespace of the WSDL 2.0 document.
	When such absolute URIs and IRIs are being compared to determine equivalence (see § 2.15 – Equivalence of Components on page 62), they MUST be compared character-by-character as indicated in [IETF RFC 3987].
	The value of the targetNamespace *attribute information item* SHOULD be dereferencable.
	It SHOULD resolve to a human or machine processable document that directly or indirectly defines the intended semantics of those components.
	It MAY resolve to a WSDL 2.0 document that provides service description information for that namespace.
	For each component in the imported namespace, a corresponding Element Declaration component or Type Definition component MUST appear in the element declarations or type definitions property respectively of the Description component corresponding to the WSDL document that imports the schema, or that imports directly or indirectly a WSDL document that imports the schema.
	Schema components not in an imported namespace MUST NOT appear in the element declarations or type definitions properties.
	For each component defined and declared in the inlined schema document or included by xs:include, a corresponding Element Declaration component or Type Definition component MUST appear in the element declarations property or type definitions property respectively of the Description component corresponding to the WSDL document that contains the schema, or that imports directly or indirectly a WSDL document that contains the schema.
	Schema components not defined or declared in the inlined schema document or included by xs:include MUST NOT appear in the element declarations or type definitions properties.
	This *xs:anyURI* MUST be an absolute IRI as defined by [IETF RFC 3987].
	For each Endpoint component in the endpoints property of a Service component, the binding property MUST either be a Binding component with an unspecified interface property or a Binding component with an interface property equal to the interface property of the Service component.
	Extension properties which are not string values, sets of strings or references MUST describe their values' equivalence rules.
	An extension that is NOT marked as mandatory MUST NOT invalidate the meaning of any part of a 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.
	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.
	The meaning of an extension SHOULD be defined (directly or indirectly) in a document that is available at its namespace IRI.
	For QNames, any prefix MUST be defined by a preceding xmlns pointer part.

	The fragment identifier in a WSDL 2.0 component IRI-reference MUST resolve to some component as defined by the construction rules in Rules for determining pointer parts for WSDL 2.0 components Component 1 2 3 4 Pointer Part Description n/a n/a n/a n/a wsdl.description() Element Declaration element QName n/a n/a n/a wsdl.elementDeclaration(element) Element Declaration element QName system IRI n/a n/a wsdl.elementDeclaration(element,system) Type Definition type QName n/a n/a n/a wsdl.typeDefinition(type) Type Definition type QName system IRI n/a n/a wsdl.typeDefinition(type,system) Interface interface NCName n/a n/a n/a wsdl.interface(interface) Interface Fault interface NCName fault NCName n/a n/a wsdl.interfaceFault(interface/fault) Interface Operation interface NCName operation NCName n/a n/a wsdl.interfaceOperation(interface/operation) Interface Message Reference interface NCName operation NCName message NCName n/a wsdl.interfaceMessageReference(interface/operation/message) Interface Fault Reference interface NCName operation NCName message NCName fault QName wsdl.interfaceFaultReference(interface/operation/message/fault) Binding binding NCName n/a n/a n/a wsdl.binding(binding) Binding Fault binding NCName fault QName n/a n/a wsdl.bindingFault(binding/fault) Binding Operation binding NCName operation QName n/a n/a wsdl.bindingOperation(binding/operation) Binding Message Reference binding NCName operation QName message NCName n/a wsdl.bindingMessageReference(binding/operation/message) Binding Fault Reference binding NCName operation QName message NCName fault QName wsdl.bindingFaultReference(binding/operation/message/fault) Service service NCName n/a n/a n/a wsdl.service(service) Endpoint service NCName endpoint NCName n/a n/a wsdl.endpoint(service/endpoint) Extensions namespace URI identifier extension-specific-syntax n/a n/a wsdl.extension(namespace,identifier) .
	The semantics of an extension MUST NOT depend on how components are brought into a component model instance via <import> or <include>.
	To avoid circular definitions, an interface MUST NOT appear in the set of interfaces it extends, either directly or indirectly.
	For each Interface component in the interfaces property of a Description component, the name property MUST be unique.
	The list of *xs:QName* in an extends *attribute information item* MUST NOT contain duplicates.
	An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.
	When the message content model property has the value *#any* or *#none* the element declaration property MUST be empty.
	In cases where, due to an interface extending one or more other interfaces, two or more Interface Fault components have the same value for their name property, then the component models of those Interface Fault components MUST be equivalent (see § 2.15 – Equivalence of Components on page 62).
	For the above reason, it is considered good practice to ensure, where necessary, that the local name of the name property of Interface Fault components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error.
	The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
	The direction MUST be consistent with the direction implied by the fault propagation ruleset used in the message exchange pattern of the operation.
	For each Interface Fault Reference component in the interface fault references property of an Interface Operation component, the combination of its interface fault and message label properties MUST be unique.
	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.
	The direction MUST be the same as the direction of the message identified by the message label property in the message exchange pattern of the Interface Operation component this is contained within.
	An *xs:token* with one of the values *#any*, *#none*, *#other*, or *#element*.
	When the message content model property has the value *#any* or *#none*, the element declaration property MUST be empty.
	For each Interface Message Reference component in the interface message references property of an Interface Operation component, its message label property MUST be unique.
	This *xs:anyURI* MUST be an absolute IRI (see [IETF RFC 3987]).

	These *xs:anyURI*s MUST be absolute IRIs (see [IETF RFC 3986]).
	In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation components have the same value for their name property, then the component models of those Interface Operation components MUST be equivalent (see § 2.15 – Equivalence of Components on page 62).
	For the above reason, it is considered good practice to ensure, where necessary, that the name property of Interface Operation components within a namespace SHOULD be unique, thus allowing such derivation to occur without inadvertent error.
	An Interface Operation component MUST satisfy the specification defined by each operation style identified by its style property.
	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], 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]) for that namespace name.
	A message exchange pattern is itself uniquely identified by an absolute IRI, which is used as the value of the message exchange pattern property of the Interface Operation component, and which specifies the fault propagation ruleset that its faults obey.
	The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
	For each Service component in the services property of a Description component, the name property MUST be unique.
	Each XML Schema element declaration MUST have a unique QName.
	Each XML Schema type definition MUST have a unique QName.

This page is intentionally left blank.