Date: May 2010

CORBA Binding for WSDL

Version 1.0

OMG Document Number: formal/2010-05-08
Standard document URL: http://www.omg.org/spec/CORBABINDING/1.0/

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

IONA Technologies, PLC

The companies listed above have granted to the Object Management Group, Inc. (OMG) a non-exclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-pad up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page *http://www.omg.org*, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface iii

1 Scope 1

2 Conformance 1

3 Normative References 1

4 Terms and Definitions 2

5 Symbols 2

6 Additional Information 2

6.1 Changes to Adopted OMG Specifications 2

6.2 How to Read this Specification 2

6.3 Acknowledgements 3

6.4 Proof of Concept 3

7 CORBA Binding for WSDL 5

7.1 Overview 5

7.2 Namespace 5

7.3 Mapping IDL to XML Schema 5

7.4 CORBA Type Map 6

7.4.1 Primitives 6
7.4.2 Constant 7
7.4.3 Enum 7
7.4.4 Struct 8
7.4.5 Exception 9
7.4.6 Fixed 10
7.4.7 Union 11
7.4.8 Typedef 12
7.4.9 Bounded/Unbounded Strings 13
7.4.10 Array 13
7.4.11 Sequence 14
7.4.12 Anonymous types 15
7.4.13 String 15
7.4.14 Fixed 15
7.4.15 Sequence 16
7.4.16 Array 16
7.4.17 Object References 17

7.5 CORBA Binding 18
7.5.1 Binding Element 18
7.6 CORBA Services 20
7.7 CORBA Types Not Supported 21

A - Translation from IDL to WSDL/XMLSchema 23
Index 25

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices

- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 Scope

This specification defines a CORBA binding for WSDL (Web Services Description Language) to allow CORBA services to be described using WSDL, and to allow native CORBA communication mechanisms to be specified in WSDL.

To achieve an unambiguous, bijective mapping between CORBA and WSDL/XMLSchema, this specification defines:

- A mapping of IDL types to XML Schema types, based on the "CORBA to WSDL/SOAP Interworking Specification" (formal/06-11-01), with the exceptions detailed in Section 7.3, "Mapping IDL to XML Schema" of this specification.
- An extension to the WSDL `definitions` element that defines a CORBA type map element. The purpose of the CORBA type map is to retain the information lost in translation from IDL definitions to WSDL/XMLSchema definitions (see "Annex A" on page 23).
- Extensions to the WSDL physical elements `binding` and `service`, to define CORBA binding and CORBA service elements.

Compliance to this specification will enable:

- Existing CORBA endpoints to be exposed as Web services, enabling Web service applications access to existing CORBA services.
- Existing Web services to respond to existing CORBA clients invocations by simply adding a CORBA `typemap` element, a CORBA `binding` element, and a CORBA `service` element in their WSDL contracts.
- Existing Web services clients and servers to use CORBA IIOP as an underlying transport by simply modifying their WSDL contracts.

2 Conformance

Implementations must support the entire mapping, specifically those outlined in Chapter 7.

3 Normative References

- WSDL 1.1: "Web Services Description Language (WSDL) 1.1" *http://www.w3.org/TR/wsdl*
- XML Schema 1.0: "XML Schema Part 1: Structures" *http://www.w3.org/TR/xmlschema-1/*, "XML Schema Part 2: Datatypes" *http://www.w3.org/TR/xmlschema-2/*
- XML Schema 1.1: "XML Schema 1.1 Part 1: Structures" (Working Draft) *http://www.w3.org/TR/2004/WD-xmlschema11-1-20040716/*, "XML Schema 1.1 Part 2: Datatypes" (Working Draft) *http://www.w3.org/TR/2004/WD-xmlschema11-2-20040716/*
- Xpath 1.0: "XML Path Language (XPath) Version 1.0" *http://www.w3.org/TR/xpath*
- WSI-BP 1.0: "Web Services Interoperability Basic Profile Version 1.0" *http://www.ws-i.org/Profiles/BasicProfile-1.0- 2004-04-16.html*

- OMG IDL Syntax and Semantics defined in CORBA 2.6 Specification (formal/01-12-35) *http://www.omg.org/cgi-bin/doc?formal/01-12-35*
- CORBA to WSDL/SOAP Interworking 1.2.1 (formal/2008-08-03) *http://www.omg.org/cgi-bin/doc?formal/2008-08-03*
- Web Services Addressing (WS-Addressing) *http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/*

4 Terms and Definitions

This specification defines no new terms other than those defined by those documents listed in the Normative References section (3).

5 Symbols

List of symbols/abbreviations.

IDL—Interface Definition Language

WSDL—Web Services Description Language

CORBA—Common Object Request Broker Architecture

SOAP—Simple Object Access Protocol

6 Additional Information

6.1 Changes to Adopted OMG Specifications

The changes made by this CORBA Binding for WSDL document in Section 7.3 to the existing CORBA to WSDL/SOAP interworking specification need to be reflected in a new version of CORBA to WSDL/SOAP interworking specification.

This new version also needs to include the specification of a new version number to include in the mapping version indicator schema definition.

6.2 How to Read this Specification

The rest of this document contains the technical content of this specification. This document provides a specification for the mapping of IDL to WSDL, and with a few exceptions, which are noted in the specification itself, is complete. In addition, the mapping, as described in this specification, is already available to the market as a product.

6.3 Acknowledgements

The following companies submitted and/or supported parts of this specification:

- IONA Technologies
- Fujitsu Software Corp. (supporter)

6.4 Proof of Concept

IONA Technologies PLC has completed a product in the market, which is the basis of this specification. This product is part of a suite of middleware interoperability design and implementation tools known collectively as Artix™. IONA Technologies PLC is also involved in the Apache CXF open source project to build a robust open source services framework and part of its runtime and tooling components are based on this specification.

7 CORBA Binding for WSDL

7.1 Overview

Describing a CORBA object in WSDL essentially requires recasting both its IDL definition and its communication details as WSDL definitions. Specifically, this requires:

1. Mapping IDL types used by the CORBA object's interface to XML schema types.
2. Defining the operation and attribute details of the object's interface as WSDL `portType` and `binding` elements.
3. Defining a WSDL `service` that combines the type and interface information defined in the prior steps with the communication details from the object's reference.

This specification, which targets CORBA 2.6 and WSDL 1.1, details an approach for fulfilling these fundamental requirements.

This specification details an approach for fulfilling the following requirements. The specification supports CORBA 2.6 with the exception of types specified in Section 7.7, 'CORBA Types Not Supported' and supports WSDL 1.1.

Text in files included with a #include directive are treated as if it appeared in the including file.

7.2 Namespace

CORBA bindings require the use of the `corba` namespace, defined as follows:

`xmlns:corba="`*`TO_BE_ASSIGNED_BY_OMG`*`"`

7.3 Mapping IDL to XML Schema

This specification follows the IDL to XML Schema mapping defined in the "CORBA to WSDL/SOAP Interworking Specification" (formal/06-11-01), except for the following differences (note that XML Schema types in the following list use the common namespace prefix "xs" to distinguish them from IDL types):

- IDL `any` maps to `xs:anyType`. While the IDL type comprises a `TypeCode` and a value, what matters in most applications is just the value, given that `TypeCode` does not map well into non-CORBA applications.
- IDL `sequence<octet>` maps to either `xs:base64Binary` or `xs:hexBinary`.
- IDL `sequence<octet,n>` maps to a restricted `xs:simpleType` based on either `xs:base64Binary` or `xs:hexBinary`. The `xs:maxLength` facet's value is `n`.

```
typedef sequence<octet,10> OctetSeq;

<xs:simpleType name="OctetSeq">
   <xs:restriction base="xs:base64Binary">
      <xs:maxLength value="10"/>
   <xs:restriction>
<xs:simpleType>
```

- IDL `octet[n]` maps to a restricted `xs:simpleType` based on either `xs:base64Binary` or `xs:hexBinary`. The `xs:length` facet's value is `n`.

```
typedef octet[10] OctetArray;

<xs:simpleType name="OctetArray">
   <xs:restriction base="xs:base64Binary">
      <xs:length value="10"/>
   <xs:restriction>
<xs:simpleType>
```

In this specification, the original mapping of modules is preserved (i.e., they are mapped as prefixes separated by '.' characters, e.g., module `M` containing definition `A` results in the name `M.A` for that definition).

7.4 CORBA Type Map

Because of the impedance mismatch between XML Schema and IDL, it is impossible to fully recover the original definition of a CORBA type from just its mapped XML Schema type alone. This specification therefore specifies a WSDL extension called a "CORBA type map." The type map specifies CORBA type definitions that are used within CORBA bindings to accurately specify constants, parameter types, return types, and exception types.

The CORBA type map is specified in a WSDL element that appears as a child of the WSDL definitions element. Because IDL types are not globally unique, the CORBA type map element requires a targetNamespace attribute that should be unique and is represented by a URI. Namespaces should conform to the specification at: *http://www.w3.org/TR/1999/REC-xml-names-19990114/*.

For example:

```
<corba:typeMapping targetNamespace="http://example.com/idl/example/typemap">
</corba:typeMapping>
```

Examples in this specification that refer to the CORBA type map namespace use the namespace prefix `corbatm`.

7.4.1 Primitives

The primitive mappings are shown in the table below.

Table 7.1 - Schema mapping for CORBA primitive types

IDL type	CORBA type	XML Schema type
`long`	`corba:long`	`xs:int`
`unsigned Long`	`corba:ulong`	`xs:unsignedInt`
`long long`	`corba:longlong`	`xs:long`
`unsigned long long`	`corba:ulonglong`	`xs:unsignedLong`
`short`	`corba:short`	`xs:short`
`unsigned short`	`corba:ushort`	`xs:unsignedShort`
`float`	`corba:float`	`xs:float`

Table 7.1 - Schema mapping for CORBA primitive types

IDL type	CORBA type	XML Schema type
double	corba:double	xs:double
long double	corba:longdouble	xs:double
char	corba:char	xs:byte
wchar	corba:wchar	xs:string
boolean	corba:boolean	xs:boolean
Octet	corba:octet	xs:unsignedByte
string	corba:string	xs:string
wstring	corba:wstring	xs:string
any	corba:any	xs:anyType

7.4.2 Constant

An IDL constant is specified by a `corba:const` element. It has four required attributes:

1. `name`: the fully-qualified name of the constant
2. `value`: the value of the constant
3. `idltype`: the IDL type of the constant
4. `type`: the Schema type of the constant

For example, the following IDL constant definition:

```
// IDL
const short Length = 5;
```

results in the following type map definition:

```
<corba:typeMapping targetNamespace="http://mycompany.com/myidl">
    <corba:const name="Length" value="5" idltype="corba:short" type="xs:short"/>
</corba:typeMapping>
```

7.4.3 Enum

An IDL enum is defined by a `corba:enum` element. This element has three required attributes, as listed below, and has one child `corba:enumerator` element for each enumerator. A `corba:enumerator` element has a single attribute, the value of the enumerator.

1. `name`: fully-scoped name of the `enum` type
2. `repositoryID`: the repository ID of the `enum` type
3. `type`: the Schema type of the `enum` type

For example, the following `enum` definition:

```
// IDL
module M {
   enum Color { RED, BLUE, GRAY };
};
```

results in the following type map definition:

```
<corba:typeMapping targetNamespace="http://mycompany.com/myidl">
    <corba:enum name="M.Color" repositoryID="IDL:M/Color:1.0" type="xsd1:M.Colour">
       <corba:enumeration value="RED"/>
       <corba:enumeration value="BLUE"/>
       <corba:enumeration value="GRAY"/>
    </corba:enum>
</corba:typeMapping>
```

7.4.4 Struct

An IDL `struct` is specified by a `corba:struct` element. This element has three required attributes, as listed below, and has one child `corba:member` element for each `struct` member. A `corba:member` element requires two attributes, the `name` of the member and its `idltype`.

1. `name`: the fully-qualified name of the `struct` type
2. `repositoryID`: the repository ID of the `struct` type
3. `type`: the Schema type of the `struct` type

For example, the following `struct` definition:

```
// IDL
struct Employee {
   string name;
   long id;
};
```

results in the following XMLSchema definition, in accordance with the “CORBA to WSDL/SOAP Interworking Specification” - formal/06-11-01 (this example is defined in a namespace identified by the `xsd1` prefix):

```
<xs:complexType name="Employee">
    <xs:sequence>
        <xs:element name="name" type="xs:string">
        </xs:element>
        <xs:element name="id" type="xs:int">
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

and results in the following type map definition:

```
<corba:struct repositoryID="IDL:Employee:1.0" type="xsd1:Employee" name="Employee">
   <corba:member name="name" idltype="corba:string" />
   <corba:member name="id" idltype="corba:long" />
</corba:struct>
```

And the following `struct` definition, which reuses previously defined `struct Employee` type:

```
// IDL
struct EmployeeDepartment {
  Employee emp;
  string   dept;
};
```

results in the following type map definition:

```
<corba:struct repositoryID="IDL:EmployeeDepartment:1.0" type="xsd1:EmployeeDepartment"
   name="EmployeeDepartment">
   <corba:member name="emp" idltype="corbatm:Employee" />
   <corba:member name="dept" idltype="corba:string" />
</corba:struct>
```

7.4.5 Exception

Both a User or a System IDL exception is specified by a `corba:exception` element, which is identical to the `corba:struct` element except that it can be empty, i.e., it can hold zero or more `corba:member` child elements.

This element has three required attributes, as listed below, and if members exist it will have one child `corba:member` element for each `struct` member. A `corba:member` element requires two attributes, the name of the member and its `idltype`.

1. `name`: the fully-qualified name of the `struct` type
2. `repositoryID`: the repository ID of the `struct` type
3. `type`: the Schema type of the `struct` type

For example, the following `exception` definition with no members:

```
// IDL
Exception NotFound {};
```

results in the following type map definition:

```
<corba:exception name="NotFound" repositoryID="IDL:NotFound:1.0" type="xsd1:NotFound">
</corba:exception>
```

And, the following `exception` definition with members:

```
// IDL
exception NotFound {
   string reason;
   string type;
};
```

results in the following type map definition:

```
<corba:exception name="NotFound" repositoryID="IDL:NotFound:1.0" type="xsd1:NotFound">
   <corba:member name="reason" idltype="corba:string"/>
   <corba:member name="type" idltype="corba:string"/>
</corba:exception>
```

And the following exception definition, which reuses the previously defined `struct Employee` type:

```
// IDL
exception NotFound {
  Employee emp;
  string reason;
};
```

results in the following type map definition:

```
<corba:exception repositoryID="IDL:Foo/NotFound:1.0" type="xsd1:NotFound" name="NotFound">
   <corba:member name="emp" idltype="corbatm:Employee" />
   <corba:member name="reason" idltype="corba:string" />
</corba:exception>
```

7.4.6 Fixed

An IDL fixed-point type is specified by a `corba:fixed` element. This element has five required attributes:

1. `name`: the name of the fixed-point type
2. `repositoryID`: the repository ID of the fixed-point type
3. `digits`: the number of digits specified for the fixed-point type
4. `scale`: the scale of the fixed-point type
5. `type`: the Schema type of the fixed-point type

For example, the following fixed-point type definition:

```
// IDL
typedef fixed<4,2> MyFixed;
```

results in the following type map definition:

```
<corba:fixed name="MyFixed" repositoryID="IDL:MyFixed:1.0" digits="4" scale="2"
    type="xsd:decimal"/>
```

7.4.7 Union

An IDL `union` type is specified by a `corba:union` element. This element has four required attributes and one optional attribute, as listed below, and has one child `corba:unionbranch` element for each `union` member. A `corba:unionbranch` element requires two attributes, the `name` of the `union` member and its `idltype`. It also contains one child `corba:case` element for each discriminator value that corresponds to this `union` member. The discriminator value is specified by the `label` attribute.

1. `name`: the name of the `union` type
2. `repositoryID`: the repository ID of the `union` type
3. `discriminator`: the type of the discriminator of the `union` type
4. `default`: this optional `boolean` attribute, which defaults to `false`, is set to `true` for the default branch of the `union`, if any
5. `type`: the Schema type of the `union` type

For example, the following `union` definition:

```
// IDL
union MyUnion switch(short) {
   case 0:
      string case0;
   case 1:
   case 2:
      float case12;
   case 3:
      Employee case3;
   default:
      long case_def;
};
```

results in the following type map definition:

```
<corba:union name="MyUnion" repositoryID="IDL:MyUnion:1.0" discriminator="corba:short"
   type="xsd1:MyUnion">
   <corba:unionbranch name="case0" idltype="corba:string">
      <corba:case label="0"/>
   </corba:unionbranch>
   <corba:unionbranch name="case12" idltype="corba:float">
      <corba:case label="1"/>
       <corba:case label="2"/>
   </corba:unionbranch>
   <corba:unionbranch name="case3" idltype="corbatm:Employee">
       <corba:case label="3"/>
   </corba:unionbranch>
   <corba:unionbranch name="case_def" idltype="corba:long" default="true"/>
</corba:union>
```

7.4.8 Typedef

An IDL `typedef` is specified by a `corba:alias` element. This element has four required attributes:

1. `name`: the name of the `typedef`
2. `repositoryID`: the repository ID of the `typedef`
3. `basetype`: the IDL base type
4. `type`: the Schema type

For example, the following typedef definition:

```
// IDL
typedef long MyLong;
```

results in the following type map definition:

```
<corba:alias name="MyLong" repositoryID="IDL:MyLong:1.0" basetype="corba:long"
   type="xsd:int"/>
```

And the following typedef definition:

```
// IDL
typedef Employee MyEmployee;
```

results in the following type map definition:

```
<corba:alias basetype="corbatm:Employee" repositoryID="IDL:MyEmployee:1.0"
name="MyEmployee" />
```

7.4.9 Bounded/Unbounded Strings

An IDL unbounded string maps to a primitive `xsd:string` as previously indicated.

An IDL bounded string is specified by a `corba:anonstring`.

The `corba:anonstring` has three required attributes:

1. `name`: the name of the `string`
2. `bound`: the bound of the `string`
3. `type`: the Schema type

For example:

```
// IDL
typedef string <10> myString;
```

results in the following type map definition:

```
<corba:anonstring bound="10" name="_1_myString" type="xsd1:myString"/>
<corba:alias basetype="corbatm:_1_myString" name="mystring" repositoryID="IDL:myString:1.0"
   type="xsd1:myString"/>
```

7.4.10 Array

An IDL array is specified by a `corba:array` element. This element has five required attributes:

1. `name`: the name of the array
2. `repositoryID`: the repository ID of the array type
3. `elemtype`: the element type of the array
4. `bound`: the size of the array
5. `type`: the Schema type of the array

For example, the following array definition:

```
// IDL
typedef long MyArray[5];
```

results in the following type map definition:

```
<corba:array name="MyArray" repositoryID="IDL:MyArray:1.0" elemtype="corba:long"
     bound="5" type="xds1:MyArray"/>
```

And the following array definition:

```
// IDL
typedef Employee MyEmployees[10];
```

results in the following type map definition:

```
<corba:array elemtype="corbatm:Employee" elemname="item" bound="10"
   repositoryID="IDL:MyEmployees:1.0" type="xsd1:MyEmployees" name="MyEmployees" />
```

7.4.11 Sequence

An IDL sequence is specified by a `corba:sequence` element. This element has five required attributes:

1. `name`: the name of the sequence
2. `repositoryID`: the repository ID of the sequence type
3. `elemtype`: the element type of the sequence type
4. `bound`: the maximum size of the sequence type. An unbounded sequence has a bound value of zero.
5. `type`: the Schema type of the sequence type

For example, the following sequence definitions:

```
// IDL
typedef sequence<long> MySeq;
typedef sequence<string, 10> MyBStringSeq;
```

result in the following type map definition:

```
<corba:sequence name="MySeq" repositoryID="IDL:MySeq:1.0" elemtype="corba:long"
   bound="0" type="xsd1:MySeq"/>
<corba:sequence name="MyBStringSeq" repositoryID="IDL:MyBStringSeq:1.0"
   elemtype="corba:string" bound="10" type="xsd1:MyBStringSeq"/>
```

And the following sequence definitions:

```
// IDL
typedef sequence<Employee> MyEmpSeq;
typedef sequence<Employee, 20> MyBoundedEmpSeq;
```

result in the following type map definition:

```
<corba:sequence elemtype="corbatm:Employee" elemname="item" bound="0"
   repositoryID="IDL:MyEmpSeq:1.0" type="xsd1:MyEmpSeq" name="MyEmpSeq" />
<corba:sequence elemtype="corbatm:Employee" elemname="item" bound="20"
   repositoryID="IDL:MyBoundedEmpSeq:1.0" type="xsd1:MyBoundedEmpSeq"
   name="MyBoundedEmpSeq" />
```

7.4.12 Anonymous types

Though their use is now deprecated, some older IDL definitions still use anonymous types. In such cases, special elements are used in the CORBA type map to specify the anonymous types. All anonymous type elements have at least the following required attribute:

- `name`: name of the anonymous type. This name is used only to refer to the type within the type map, so while the precise form of the name is not standardized, it must be guaranteed to be unique within the type map.

Specific anonymous type elements may require additional attributes or child elements as well.

7.4.12.1 String

The `corba:anonstring` and `corba:anonwstring` elements are used to specify anonymous `string` and `wstring` types, respectively. Each element requires three attributes: name as described above, and bound, which specifies the bound of the string. Unbounded strings have a bound of zero. The type attribute specifies the schema type.

7.4.12.2 Fixed

The `corba:anonfixed` element is used to specify anonymous fixed-point types. This element requires four attributes: `name` as described above, as well as `digits`, which specifies the number of digits for the fixed-point type, and `scale`, which defines the scale of the fixed-point type. The type attribute specifies the schema type.

For example, the following struct definition containing an anonymous string and anonymous fixed-point type:

```
// IDL
struct S {
   string<4> str;
   fixed<5,3> fx;
};
```

results in the following type map:

```
<corba:anonstring name="_1_S" bound="4" type="xsd1:_1_S"/>
<corba:anonfixed name="_2_S" digits="5" scale="3" type="xsd:decimal"/>
<corba:struct name="S" repositoryID="IDL:S:1.0" type="xsd1:s">
   <corba:member name="str" idltype="corbatm:_1_S"/>
   <corba:member name="fx" idltype="corbatm:_2_S"/>
</corba:struct>
```

7.4.12.3 Sequence

The corba:anonsequence is deprecated since Corba 2.6. The corba:anonsequence element is used to specify anonymous sequences. This element requires four attributes: name as described above, as well as elemtype, which defines the type of the sequence element type, and bound, which specifies the maximum size of the sequence. The type attribute specifies the schema type.

For example, the following sequence definition:

```
// IDL
typedef sequence<sequence<long> > SeqSeqLong;
```

results in the following type map:

```
<corba:anonsequence name="-2-SeqSeqLong" elemtype="corba:long" bound="0"
   type="xsd1:_2_SeqSeq:Long"/>
<corba:sequence name="SeqSeqLong" repositoryID="IDL:SeqSeqLong:1.0"
   elemtype="corbatm:_2_SeqSeqLong" bound="0" type="xsd1:SeqSeqLong"/>
```

7.4.12.4 Array

The corba:anonarray element is used to specify anonymous arrays. This element requires four attributes: name as described above, as well as elemtype, which defines the type of the array element type, and bound, which specifies the size of the array. The type attribute specifies the schema type.

For example, the following multidimensional array definition:

```
// IDL
typedef long MyArray[5][10];
```

results in the following type map definition:

```
<corba:anonarray name="-1-MyArray" elemtype="corba:long" bound="5"
   type="xsd1:_1_MyArray"/>
<corba:array name="MyArray" repositoryID="IDL:MyArray:1.0"
   elemtype="corbatm:_1_MyArray" bound="10" type="xsd1:MyArray"/>
```

The corba:anonarray element is also used for cases in which array members of constructed types, such as structs, are defined using anonymous array types.

7.4.13 Object References

Object references in IDL can be passed as a parameter or a return value of an operation.

An IDL `Object Reference` is specified by a `corba:object` element. The `corba:object` represents either a generic object reference of the built in type "Object" or a type specific object reference where the IDL type is a custom specific type.

This element has four required attributes:

1. `name`: the name of type.
2. `repositoryID`: the repository ID of the `Object Reference` type.
3. `binding`: the name of the WSDL binding element associated to the CORBA binding of the type, as defined in section 7.5. For the generic object reference case this will be left blank.
4. `type`: `wsa:EndpointReferenceType`

The IDL built-in type `Object` maps to the `wsa:EndpointReferenceType` in WSDL. It is defined by the WS-Addressing Standard. So the following will need to be added to definitions section in the WSDL contract:

```
<import namespace="http://www.w3.org/2005/08/addressing" schemaLocation="WSAddressingURL">
```

where `WSAddressingURL` can either be the path to an `.xsd` file on your local file system or a URL to retrieve the schema from a remote location.

Example: Built in type Object

```
// IDL
interface Bank {
   Object create_account(in string account_name);
};
```

In order to declare the endpoint reference type as the return value from the `create_account` operation, the operation's request and reply messages should be as follows:

```
<message name="create_account">
   <part name="account_name" type="xsd:string"/>
</message>
<message name="create_accountResponse">
   <part name="return" type="wsa:EndpointReferenceType"/>
</message>
```

This results in the following generic object reference in the type map definition:

```
<corba:object binding="" name="CORBA.Object" repositoryID="IDL:omg.org/CORBA/Object/1.0"
   type="wsa:EndpointReferenceType"/>
```

Example: An operation returns a reference to a specific type. The return value is defined to be of Account type.

```
// IDL
interface Account {
    float get_balance();
};

interface Bank {
  ::Account create_account(in string account_name);
};
```

The operation's request and reply message should be as follows.

```
<message name="create_account">
    <part name="account_name" type="xsd:string"/>
</message>
<message name="create_accountResponse">
    <part name="return" type="wsa:EndpointReferenceType"/>
</message>
```

This results in the following type-specific object reference in the type map definition:

```
<corba:object binding="AccountCORBABinding" name="Account"
   repositoryID="IDL:Account:1.0" type="wsa:EndpointReferenceType"/>
```

7.5 CORBA Binding

This specification extends WSDL with a CORBA-specific binding. Such a binding pertains to a specific IDL interface, which we refer to as the *primary interface* below.

Examples in this section that specify WSDL entities use the common namespace prefix "wsdl" where appropriate to distinguish them from CORBA definitions.

7.5.1 Binding Element

A `corba:binding` element appears within a WSDL binding element. It has one required attribute named `repositoryID`. This attribute specifies the repository ID of the primary interface. It also has an optional attribute named `bases` that specifies the repository IDs of the base interfaces of the primary interface. The `bases` attribute can be used to look up operations and attributes inherited by the primary interface.

For example, the `corba:binding` element for interface `Foo` with bases `Base1` and `Base2`, assuming default repository IDs for all three interfaces, is specified as follows:

```
<wsdl:binding name="FooCORBABinding" type="tns:Foo">
    <corba:binding repositoryID="IDL:Foo:1.0" bases="IDL:Base1:1.0 IDL:Base2:1.0"/>
</wsdl:binding>
```

A corba:operation element, which appears within a WSDL operation element, defines an IDL operation. It has one required attribute named name, which specified the operation name. A corba:operation element holds zero or more corba:param elements, which are used to specify the operation's parameters. Operations with a non-void return value also specify a corba:return element, which specifies the operation's return type and name. The name attribute of a corba:param element specifies the parameter name. Its mode attribute specifies the direction of the parameter, and must be one of "in," "inout," or "out." Its idltype attribute specifies the IDL type of the parameter. The corba:return element also has an idltype attribute, which specifies the return type of the operation.

For example, given the Foo interface used in the previous example:

```
// IDL
interface Foo : Base1, Base2
{
    string lookup(in string x);
};
```

The corba:binding for Foo, including its operation definition, is specified as follows:

```
<wsdl:binding name="FooCORBABinding" type="ns:Foo">
   <corba:binding repositoryID="IDL:Foo:1.0" bases="IDL:Base1:1.0 IDL:Base2:1.0"/>
   <wsdl:operation name="lookup">
      <corba:operation name="lookup">
         <corba:param name="x" mode="in" idltype="corba:string"/>
         <corba:return idltype="corba:string" name="return"/>
      </corba:operation>
      <wsdl:input/>
      <wsdl:output/>
    </wsdl:operation>
</wsdl:binding>
```

IDL attributes are treated as two operations, named by preceding the attribute name with _get_ to define the read operation and _set_ to define the write operation. Readonly attributes have only the _get_ operation.

All exceptions appearing within an operation's raises clause are specified using the corba:raises element. This element has one required attribute, exception, which specifies the exception type from the CORBA type map. A separate corba:raises element appears for each exception in the raises clause.

For example, adding an exception to the Foo interface used in the previous example:

```
// IDL
interface Foo : Base1, Base2
{
    exception NotFound {};
    string lookup(in string x) raises(NotFound);
};
```

The corba:binding for Foo is specified as follows:

```
<wsdl:binding name="FooCORBABinding" type="ns:Foo">
   <corba:binding repositoryID="IDL:Foo:1.0" bases="IDL:Base1:1.0 IDL:Base2:1.0"/>
   <wsdl:operation name="lookup">
      <corba:operation name="lookup">
         <corba:param name="x" mode="in" idltype="corba:string"/>
         <corba:return idltype="corba:string"/>
         <corba:raises exception="corbatm:Foo.NotFound"/>
      </corba:operation>
      <wsdl:input/>
      <wsdl:output/>
      <wsdl:fault name="Foo.NotFound"/>
    </wsdl:operation>
</wsdl:binding>
```

Note that in the example above, the type of the exception in the corba:raises clause refers to the Foo.NotFound definition specified in the accompanying CORBA type map (not shown) and also maps to the wsdl:fault element within the binding.

7.6 CORBA Services

Defining a WSDL service with a CORBA binding requires specifying the CORBA binding as the service binding, and supplying the location of a CORBA object that can fulfill that binding.

A CORBA object location is specified as the value of the location attribute of a corba:address element. The location can be defined in the following ways:

- A file URL that refers to a file containing a stringified object reference.
- A corbaname URL specifying an object reference within a CORBA Naming service instance o a stringified object reference.
- A corbaloc URL.
- A placeholder IOR, specified by the string “IOR:,” which allows the actual object reference to be specified by an application at runtime.

For example, given the FooCORBABinding as defined in the previous section, you can specify the filename /tmp/myobject.ior in URL form to indicate that it contains the CORBA object reference for the service:

```
<wsdl:service name="FooCORBAService">
   <wsdl:port name="FooCORBAPort" binding="tns:FooCORBABinding">
      <corba:address location="file:///tmp/myobject.ior"/>
   </wsdl:port>
</wsdl:service>
```

7.7 CORBA Types Not Supported

The following types are not supported in this specification.

Tools mapping these IDL types to WSDL should output a WARNING Message indicating that they are not supported and continue to map the remaining IDL types.

- `Value types`—difficult to implement and not widely (or properly) supported by many ORBs.
- `Boxed values`—difficult to implement and not widely (or properly) supported by many ORBs.
- `Local interfaces`—they are not remotely accessible.
- `Abstract interfaces`—difficult to implement and not widely (or properly) supported by many ORBs.

Annex A

Translation from IDL to WSDL/XMLSchema

The purpose of the CORBA type map is to retain the information lost in translation from IDL definitions to WSDL/XMLSchema definitions. For example, the “CORBA to WSDL/SOAP Interworking Specification” defines that for the following CORBA IDL type:

```
struct MyType {
    string MyString;
};
```

`struct` maps to the following XML Schema type:

```
<xs:complexType name="MyType">
  <xs:sequence>
    <xs:element name="MyString" type="xs:string">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

The following CORBA IDL type:

```
typedef string MyType[1];
```

also maps to the following XML Schema type:

```
<xs:complexType name="MyType">
  <xs:sequence>
    <xs:element name="item" type="xs:string">
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

As illustrated, two distinct IDL types (a `struct` containing a `string` and an `array` of `string` of size one) are mapped to the same XML Schema type (a `complexType` containing a sequence containing an element of type `string`).

This means that the remote end of the connection cannot determine which CORBA IDL type was sent just by looking at the information provided by the XML Schema type definitions, but requires the data encoded in the CORBA type map element.

INDEX

Symbols
#include 5

A
Abstract interfaces 21
Acknowledgements 3
Additional Information 2
Anonymous type elements 15
Array 13, 16

B
Binding 1
Binding element 18
Bounded string 13
Boxed values 21

C
Changes to Adopted OMG Specifications 2
Compliance 1
Conformance 1
Constant 7
CORBA binding 1
CORBA endpoints 1
CORBA IIOP 1
CORBA service elements 1
CORBA services 20
CORBA to WSDL/SOAP interworking specification 2
CORBA type map 6
CORBA typemap element 1
CORBA—Common Object Request Broker Architecture 2

D
Definitions 2

E
Endpoints 1
Enum 7
Exception 9

F
Fixed 15

H
How to Read this Specification 2

I
IDL—Interface Definition Language 2

L
Local interfaces 21

N
Namespace 5
Normative References 1

O
Object references 17

P
Primitive mappings 6

R
References 1

S
Scope 1
Sequence 14, 16
Service 1
SimpleTime 11
SOAP—Simple Object Access Protocol 2
String 15
Struct 8
Symbols 2

T
Terms and definitions 2
Typedef 12

U
Unbounded string 13
Union 11

V
Value types 21

W
WSDL (Web Services Description Language) 1, 2