Date: October 2003

WSDL-SOAP to CORBA Interworking

Convenience Document for FTF Final Report

ptc/03-10-17

1 Scope

This specification defines a mapping between WSDL specifications, with a SOAP Binding, to a corresponding set of OMG IDL interface specifications.

This specification is applicable to the domain of WSDL specifications which use only the constructs which result from the CORBA to WSDL-SOAP specification. This simplifies the mapping, and allows for mapping from a restricted WSDL-SOAP subset to CORBA IDL interfaces.

This specification assumes that the CORBA to WSDL-SOAP mapping includes an identifier for the source OMG IDL file in the resulting WSDL specification. The WSDL to IDL translator can key off this identifier to revert to the original IDL specification, rather than performing the translation algorithm specified in this specification.

2 Conformance

This interworking specification defines ~~three~~ four conformance points. ~~Implementations must support at least one of these conformance points:~~

Implementations must support at least one of the following three conformance points:

- Interworking between RPC/Encoded WSDL Soap Bindings and OMG IDL.
- Interworking between RPC/Literal WSDL Soap Bindings and OMG IDL.
- Interworking between Document/Literal WSDL Soap Bindings and OMG IDL.

A system that does not support a particular SOAP binding use for interaction translation (i.e., encoded vs. literal), need not translate unsupported WSDL Soap bindings.

An additional optional conformance point pertains to the reverse translation optimization, specified in section 6.2.

- An implementation may support a CORBA Client interworking with a WSDL Port, for a WSDL Port Type which was originally defined as an OMG IDL interface.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

~~Reference to CORBA - WSDL/SOAP mapping specification.~~

~~Reference to CORBA Core Chapter 3 IDL Standard.~~

~~Reference to WSDL Standard~~

~~Reference to SOAP Standard~~

~~Reference to W3C Working Draft "XML Schema Part2: Datatypes"~~

~~Reference to OMG's "Java™ Language to IDL Mapping Specification"~~

~~The FTF should complete this clause~~

CORBA to WSDL/SOAP Interworking, version FTF output, OMG Document ptc/03-05-15

Common Object Request Broker Architecture (CORBA/IIOP), version 3.0.3, OMG Document formal/2002-12-02 (Chapter 3 has IDL specification)

Java™ Language to IDL Mapping Specification, version 1.2, OMG Document formal/2002-08-06

Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, http://www.w3.org/TR/SOAP

XML Schema Part 2: Datatypes, W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2/

4 Terms and definitions

For the purposes of this specification, the terms and definitions given in the normative reference and the following apply.

~~**term**~~
~~Text of the definition~~

~~The **Terms and definitions** clause is an optional element giving definitions necessary for the understanding of certain terms used in the specification.~~

~~The term and definition list is introduced by a standard wording (above), which shall be modified as appropriate.~~

~~The FTF should complete this clause~~

~~5 Symbols~~

~~List of symbols/abbreviations~~

~~The **Symbols (and abbreviated terms)** clause is an optional element giving a list of the symbols and abbreviated terms necessary for the understanding of the standard.~~

~~Unless there is a need to list symbols in a specific order to reflect technical criteria, all symbols should be listed in alphabetical order in the following sequence:~~

- ~~upper case Latin letter followed by lower case Latin letter (*A*, *a*, *B*, *b*, etc.);~~
- ~~letters without indices preceding letters with indices, and with letter indices preceding numerical ones (*B*, *b*, *C*, C_m, C_2, *c*, *d*, d_{ext}, d_{int}, d_1, etc.);~~
- ~~Greek letters following Latin letters (*Z*, *z*, *A*, *α*, *B*, *β* … *Λ*, *λ*, etc.);~~
- ~~any other special symbols.~~

~~The FTF should complete this clause~~

The following terms are defined in CORBA/IIOP Specification:

- Interface
- Attribute
- Operation
- Module
- Exception

The following terms are defined in WSDL 1.1 Specification:

- Service
- Port
- Port type
- Message
- Binding
- Part
- Documentation
- Target namespace

The following terms are defined in SOAP 1.1 Specification:

- Soap Encoding

This specification defines no new additional terms.

6 Symbols

List of symbols/abbreviations

WSDL	Web Services Description Language
SOAP	Simple Object Access Protocol
IDL	Interface Definition Language
CORBA	Common Object Request Broker Architecture

7 WSDL to IDL Mapping

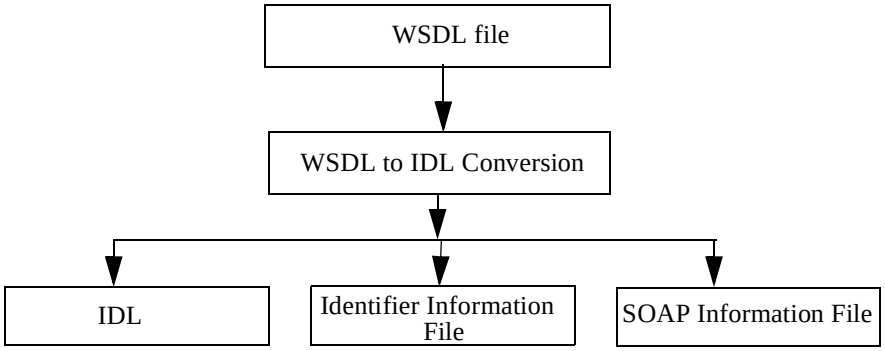
7.1 Feature Description

The overall goal of this specification is to provide a natural mapping from a valid set of WSDL service definition files to a valid set of OMG IDL specification files.

An IDL file can be generated from a WSDL file, but the generated IDL file lacks some information required to implement a CORBA/SOAP interaction translation gateway. To supplement the information, WSDL to IDL converter could (in an implementation specific manner) generate an Identifier information file and a SOAP information file in addition to the IDL file.

This specification focuses on the standard mapping of the WSDL file to the corresponding IDL specification. In

addition to the service definition, a WSDL file can have service endpoint information. This information cannot be translated into an IDL construct.



The WSDL 1.1 specification describes bindings of the following protocols.

- SOAP
- HTTP GET/POST
- MIME

However, this specification of WSDL to IDL only converts SOAP bindings. All others are considered out of scope of this specification.

In particular, this specification supports WSDL SOAP bindings with ***style*** attribute value of "***rpc***" and ***use*** attribute value of either "***literal***" or "***encoded***". In addition, it supports WSDL with ***style*** attribute value of "***document***" and ***use*** attribute value of "***literal***".

7.2 Optimization to Avoid Round-Trip Translation

If a WSDL specification is the result of translation of an OMG IDL specification, then the reverse mapping from that translated WSDL specification should be the original IDL specification.

To accomplish this, the IDL to WSDL translation specified by the CORBA to WSDL/SOAP Interworking specification provides a hint, in the form of an XML schema annotation giving both a reference to the source IDL and the version of the mapping used. Hints may be provided that refer to the source IDL file, or to the repository ID for a given generated construct (including any prefixes defined by a #pragma prefix directives).

The reverse translation from WSDL to IDL would use this hint to shortcut the translation process by having the original IDL specification be the reverse translation from the WSDL.

This will avoid round trip divergence of the IDL specifications associated with a WSDL service, which would result using the WSDL to IDL mapping translations to generate a new IDL specification associated with the WSDL output from the CORBA to WSDL-SOAP mapping specification.

~~Note – The interaction time translations must correspond to the appropriate specification translation. In particular, the interaction translation mechanisms for a service originally described in IDL are different than the interaction translation mechanisms for a service originally described in WSDL.~~

The interaction translation mechanisms required to support a CORBA Client accessing a WSDL Port differ depending on the origin of the port definition.

In particular, an implementation of this specification may encounter a WSDL port type that resulted from a translation of an IDL definition. When that happens, the interaction translation mechanisms, defined in this specification, to support a CORBA client accessing such a WSDL Port are inappropriate. The appropriate

mechanisms are similar to those required to support a CORBA server sending a response to a WSDL port for an Operation which originated as an IDL interface operation. These translation mechanisms are specified in the CORBA to WSDL/Soap specification.

Since the interaction time translation mechanisms to support this reverse specification translation option are different that those designed to support specifications originating as WSDL port types, this feature is an optional conformance point for this specification.

7.3 WSDL to IDL Conversion

IDL specification is generated from a given WSDL according to the rules shown in the following sub-sections.

7.3.1 Generation of IDL Modules

~~An IDL module is generated for each WSDL file.~~

A WSDL document may contain several different target namespaces associated with WSDL and XML constructs which are translated to corresponding IDL constructs.

A WSDL description can contain multiple namespaces, including:

- the wsdl:definitions element can have its own target namespace
- the wsdl:definition can use wsdl:import to import a wsdl namespace
- zero or more schema in the types section each have a target namespace (which is allowed, by WS-I profile, to be same as target namespace for the wsdl description
- the schema in a types section can use xsd:import to import multiple namespaces.

An IDL **module** declaration is created from the WSDL **<definitions>** element.

The WSDL <definitions> element has two optional attributes, “targetNamespace” and “name”. An IDL module is generated for a WSDL file as below:

1) If the ***targetNamespace*** attribute is present for the WSDL <definitions> element, everything before the final “/” is mapped to IDL ***typeprefix*** or ***#pragma prefix*** directive in the generated IDL file. The portion of the targetNamespace after the final “/” is mapped as the IDL ***module*** name for the generated IDL definition. Any ‘:’ character is mapped to an underscore ‘_’ character. If the generated module name contains a “.” or any special character, it is mapped to an underscore ‘_’ character.

2) If the ***targetNamespace*** attribute is not present, and the WSDL ***name*** attribute is present, the value of the WSDL ***name*** attribute is used as the IDL ***module*** name. No IDL ***typeprefix*** or ***#pragma prefix*** directive is generated in this case.

3) If neither ***targetNamespace*** nor the WSDL ***name*** attribute is present in the <***definitions***> element, then no IDL ***module*** is generated.

Using the same algorithm as specified above to map from the target namespace value to an IDL module name and type prefix, the mapping from WSDL to IDL shall use a separate IDL module for each of the target namespaces which contain constructs which are mapped to IDL constructs.

For example:

```
<!--WSDL -->
<?xml version="1.0"?>
<definitions name="StockQuote" ...>
```

```
        ...
        ...
</definitions>
```

is mapped to IDL as below:

```
// OMG IDL

module StockQuote {
        ...
        ...
};
```

The following WSDL construct maps as shown below

```
<!--WSDL -->
<?xml version="1.0"?>
<definitions name="StockQuote"
    targetNamespace="http://example.com/stockquote.wsdl"
... >
        ...
        ...
</definitions>
```

In pre-CORBA 3.0 IDL using ***pragma prefix*** as shown below:

```
// OMG IDL

#pragma prefix "http_//example.com"
module stockquote_wsdl {
        ...
        ...
};
```

Or in CORBA 3.0 or later IDL using ***typeprefix*** as shown below:

```
// OMG IDL

module stockquote_wsdl {
    typeprefix stockquote_wsdl "http_//example.com";
        ...
        ...
};
```

7.3.2 Generation of IDL Interfaces

An IDL ***interface*** declaration is created for each WSDL <***portType***> defined in the WSDL file.

The value of the ***name*** attribute of WSDL <***portType***> element is used as the IDL ***interface*** name.

For example:

```
<!--WSDL -->
<portType name="StockQuotePortType">
        ...
        ...
</portType>
```

is mapped to IDL as below:

```
// OMG IDL

interface StockQuotePortType {
```

```
        ...
        ...
};
```

7.3.3 Generation of IDL Operations

An IDL ***operation*** declaration is created for each WSDL <***operation***> element appearing inside a WSDL <***portType***> element.

The syntax of an IDL operation declaration is given below. It consists of operation name **<*op_name*>**, operation return type **<*return_type*>**, a comma seperated list of parameters **<parameter>**, and optional ***raises*** and ***context*** expression, each of which is generated from the WSDL <***operation***> element as mentioned below:

```
<return_type>  <op_name> ( <parameter> [, ])
        [ raises  ( exeption_ name [,   ] ) ]
        [ context ( context_name [,   ] ) ];
```

Operation name:

The IDL operation name **<*op_name*>** is generated from the value of the ***name*** attribute in <***operation***> element in WSDL <***portType***> declaration.

Operation type:

WSDL defines four types of operations namely, One-way, Request-response, Solicit-response and Notification. All of these are mapped to normal IDL operations. If only <***input***> message element exists in the <***portType***> element, then it is mapped to an IDL operation with return type ***void*** and with no output parameters. IDL ***oneway*** operations are not used.

Data type of return value:

The return type of the IDL ***operation*** is determined based on the following rules.

- If the Operation is of Request-Response format, the return type is the first ***part*** under the element <***wsdl:output***> if that ***part*** doesn't appear in the "***parameterOrde***r" list. Otherwise the return type is ***void***.
- If the Operation is of Solicit-Response format, the return type is the first ***part*** under the element <***wsdl:input***> if that ***part*** doesn't appear in the "***parameterOrder***" list. Otherwise the return type is ***void***.
- If it is One-Way or Notification, the return type is ***void***.

Parameters:

An IDL **<parameter>** is generated for each WSDL <***part***> element appearing inside a WSDL <***message***> element.

The syntax of the IDL operation parameter is given below. IDL supports "***in***", "***out***" and "***inout***" parameter attributes.

```
in
out     data_type  parameter_name
inout
```

IDL ***operation parameter*** generation is determined based on the following rules.

If the ***parameterOrder*** attribute is specified in the <***wsdl:operation***> element in <***portType***> declaration, parameter list is returned in the same order specified as the parts in the ***parameterOrder*** attribute. Each parameter type is determined as follows.

1) Part that is specified in both a Request/Solicit message and a Response message will be an ***"inout"*** parameter.

2) Part that is specified only in a Request/Solicit message will be an ***"in"*** parameter. One-Way, Notification will only have ***"in"*** parameters.

3) Part that is specified only in Response message will be an ***"out"*** parameter.

If the ***parameterOrder*** attribute is not specified in the <***wsdl:operation***> element in <***portType***> declaration, it is determined as follows.

1) Construct a list of all the parts in Request/Solicit/One-Way/Notification messages in the order of ***parts*** specified, excluding the first ***part*** of the Response message of Request/Solicit-Response operations.

2) Use the list as if specified for the ***parameterOrder*** attribute and process accordingly.

If the <***part***> element specifies ***"element"*** attribute instead of ***"type"***, it points to the Schema that defines the element. This becomes the ***"in"*** parameter if the '***message***' is referred by <***wsdl:input***> element in a <***wsdl:operation***> element in <***wsdl:portType***>, or the ***"out"*** parameter if the '***message***' is referred by <***wsdl:output***> element and appears in the ***parameterOrder*** attribute. If the <***wsdl:output***> element doesn't appear in the ***parameterOrder*** attribute, then it becomes the return type.

Raises expressions:

If <***wsdl:fault***> exists in an <***operation***> element inside a <***portType***> element, it is mapped to an IDL User Exception, and a ***raises*** expression is generated for the corresponding IDL operation declaration. The generated ***raises*** expression lists all the mapped user exception for that IDL operation.

The name of the generated IDL User Exception is the value of the ***name*** attribute in the <***wsdl:fault***> element. The generated User Exception structure consists of mapped data members which are list of "***parts***" that comprise the <***wsdl:message***> pointed to by the fault message name in <***wsdl:fault***> element (<***wsdl:fault name="fault_message_name"***>).

Refer to Section 7.3.6 below for an example mapping.

Context expressions:

Context expression is not generated. It is not necessary for SOAP.

The following is an example of mapping a WSDL <***operation***> element to an IDL operation.

```
<!--WSDL -->
<message name="GetTradePricesInput">
    <part name="tickerSymbol" type="xsd:string"/>
    <part name="timePeriod" type="xsd:int"/>
</message>

<message name="GetTradePricesOutput">
    <part name="result" type="xsd:string"/>
```

```
    <part name="frequency" type="xsd:float"/>
</message>

<portType name="StockQuotePortType">
    <operation name="GetTradePrices"
        parameterOrder="tickerSymbol timePeriod frequency">
        <input message="tns:GetTradePricesInput"/>
        <output message="tns:GetTradePricesOutput"/>
    </operation>
</portType>
```

The above WSDL fragment is mapped to an IDL operation as below:

```
// OMG IDL

interface StockQuotePortType {
    wstring GetTradePrices(in wstring tickerSymbol,
        in long timePeriod, out float frequency);
};
```

7.3.4 Generation of IDL Attributes

IDL attributes are not generated from WSDL.

7.3.5 Generation of IDL Typedef

An IDL typedef is generated for the XML schema type restrictions for XML schema data types used as the datatype of return values and parameters.

For example:

```
<!--WSDL -->
<xsd:simpleType name="Number">
    <xsd:restriction base="xsd:int"/>
</xsd:simpleType>

<xsd:simpleType name="AnotherNumber">
    <xsd:restriction base="Number"/>
</xsd:simpleType>
```

is mapped to IDL as below:

```
// OMG IDL

typedef long Number;
typedef Number AnotherNumber;
```

7.3.6 Generation of User Exceptions

If <***wsdl:fault***> exists in an <***operation***> element inside a <***portType***> element, it is mapped to an IDL User Exception.

The name of the generated IDL User Exception is the value of the ***name*** attribute in the <***wsdl:fault***> element. The generated User Exception structure consists of mapped data members which are list of "***parts***" that comprise the <***wsdl:message***> pointed to by the fault message name in <***wsdl:fault***> element (<***wsdl:fault name="fault_message_name"***>).

For example:

```
<!--WSDL -->
<message name="BadInput">
    <part name="errorMessage" type="xsd:string"/>
    <part name="errorCode" type="xsd:int"/>
```

```
</message>

<portType name="StockQuotePortType">
    <operation name="GetTradePrices" Ö>
        <input Ö />
        <output Ö />
        <fault message="BadInput"/>
    </operation>
</portType>
```

is mapped to IDL as below:

```
// OMG IDL

interface StockQuotePortType {

    exception BadInput {
        wstring errorMessage;
        long errorCode;
    };

    GetTradePrices( … ) raises BadInput;
};
```

7.4 Simple Type Conversion

This section shows how simple types used in WSDL are mapped to CORBA.

7.4.1 Mapping for SOAP Data Types

According to SOAP 1.1 specification, all types defined in section “3. Built-in datatypes” of “W3C Working Draft “XML Schema Part2: Datatypes” are adopted as simple types. The SOAP-ENC schema and namespace declares an element for all these simple types. Mapping of SOAP data type to IDL data type is performed according to the table below.

(In the following table, the SOAP data types are shown in the conventional “***SOAP-ENC***” namespace, to distinguish them from the IDL types).

SOAP data type	CORBA data type
SOAP-ENC:int	long
SOAP-ENC:unsignedInt	unsigned long
SOAP-ENC:short	short
SOAP-ENC:unsignedShort	unsigned short
SOAP-ENC:long	long long
SOAP-ENC:unsignedLong	unsigned long long
SOAP-ENC:float	float
SOAP-ENC:double	double
SOAP-ENC:boolean	boolean
SOAP-ENC:string	wstring The mapping for string datatype is discussed in Section 7.4.5.
SOAP-ENC:unsignedByte	octet

Enumerations .	enum Error if the base is not a string. The mapping for enumerations is discussed in Section 7.4.4
Arrays	sequence, if one-dimensional variant, array, otherwise The mapping for Array datatype is discussed in Section 7.6
Structs	struct

7.4.2 Mapping for XML Schema Built-in Datatypes

WSDL supports the XML Schema built-in datatypes which are defined in “3. Built-in datatypes” of W3C Working Draft “XML Schema Part2: Datatypes”. These XML Schema built-in datatypes maps onto a corresponding IDL type as shown in the table below.

(In the following table, the XML Schema types are shown in the conventional “***xsd***” namespace, to distinguish them from the IDL types).

XML Schema Data Type		CORBA Data Type
primitive	xsd:string	wstring The mapping for string datatype is discussed in Section 7.4.5.
	xsd:boolean	boolean
	xsd:float	float
	xsd:double	double
	xsd:decimal	See Section 7.4.3
	xsd:duration	See Section 7.4.3
	xsd:dateTime	See Section 7.4.3
	xsd:time	See Section 7.4.3
	xsd:date	See Section 7.4.3
	xsd:gYearMonth	See Section 7.4.3
	xsd:gYear	See Section 7.4.3
	xsd:gMonthDay	See Section 7.4.3
	xsd:gDay	See Section 7.4.3
	xsd:gMonth	See Section 7.4.3
	xsd:hexBinary	See Section 7.4.3
	xsd:base64Binary	See Section 7.4.3
	xsd:anyURI	wstring
	xsd:QName	See Section 7.4.3
	xsd:NOTATION	See Section 7.4.3
derived	xsd:normalizedString	wstring
	xsd:token	wstring
	xsd:language	wstring
	xsd:NMTOKEN	wstring
	xsd:NMTOKENS	wstring

	xsd:Name	wstring
	xsd:NCName	wstring
	xsd:ID	wstring
	xsd:IDREF	wstring
	xsd:IDREFS	wstring
	xsd:ENTITY	wstring
	xsd:ENTITIES	wstring
	xsd:integer	fixed
	xsd:nonPositiveInteger	See Section 7.4.3
	xsd:negativeInteger	See Section 7.4.3
	xsd:long	long long
	xsd:int	long
	xsd:short	short
	xsd:byte	See Section 7.4.3
	xsd:nonNegativeInteger	See Section 7.4.3
	xsd:unsignedLong	unsigned long long
	xsd:unsignedInt	unsigned long
	xsd:unsignedShort	unsigned short
	xsd:unsignedByte	octet
	xsd:positiveInteger	See Section 7.4.3

If the XML Schema name space is one of the following, the above conversion takes place. It will be possible to override the conversion rule with external property files.

- http://www.w3.org/2001/XMLSchema
- http://www.w3.org/2000/10/XMLSchema
- http://www.w3.org/1999/XMLSchema

7.4.3 Restriction to WSDL Type System

The following datatypes cannot be directly mapped to a corresponding OMG IDL datatype. This interworking specification provides a generic mapping of these unsupported types to individual typedefs of OMG IDL ***wstring***, to hold the UTF encoding of the XML schema type value.

decimal,

nonPositiveInteger,

nonNegativeInteger,

PositiveInteger,

NOTATION,

duration,

time,

dateTime,

date,

gYearMonth,

gYear,

gMonthDay,

gDay,

gMonth,

hexBinary,

base64Binary,

Qname,

The IDL module (using omg.org type prefix) which defines the corresponding types is:

```
module stringmappedXMLtypes {
  typeprefix stringmappedXMLtypes "omg.org";
  typedef wstring decimal;
  typedef wstring nonPositiveInteger;
  typedef wstring nonNegativeInteger;
  typedef wstring PositiveInteger;
  typedef wstring NOTATION;
  typedef wstring duration;
  typedef wstring time;
  typedef wstring dateTime;
  typedef wstring date;
  typedef wstring gYearMonth;
  typedef wstring gYear;
  typedef wstring gMonthDay;
  typedef wstring gDay;
  typedef wstring gMonth;
  typedef wstring hexBinary;
  typedef wstring base64Binary;
  typedef wstring Qname,
};
```

7.4.4 Mapping for Enumerators

The enumeration in XML Schema is used to constrain the values of almost every simple type, except the boolean type. It limits a simple type to a set of distinct values.

Enumeration in XML Schema derived by restriction on ‘***string***’ can be mapped to IDL enumeration.

Here is an example.

```
<!--WSDL -->
<simpleType name =”A_or_B_or_C” restriction base=”string”
      <enumeration value = “A” />
      <enumeration value = “B” />
      <enumeration value = “C” />
</simpleType name>
```

```
// OMG IDL

enum A_or_B_or_C {A, B, C};
```

If the restriction is on any other datatype, it cannot be mapped to OMG IDL.

7.4.5 Mapping for String Types

String is the set of finite-length sequences of characters in XML. It is mapped to OMG IDL ***wstring*** datatype.

String datatypes derived by restriction of Schema components **<*length*>** and **<*maxLength*>** are treated as bounded ***wstring***.

Note – If the value of the **<*length*>** element is 1 and value of attribute ***fixed*** is ***true***, it can be mapped to OMG IDL ***wchar*** datatype. This specification only specifies mapping to IDL ***wstring*** datatype.

String Type	WSDL	IDL
bounded	string derived by restriction of	
	• length N	wstring <N>
	• maxLength N	wstring <N>
unbounded	string	wstring
	string derived by restriction of	
	• minlength N	wstring
	• pattern	wstring

Here is an example.

```
<!--WSDL -->
<element name="Country" type="string"/>

    <element name="Place">
        <simpleType>
            <restriction base="string">
                <length value="5"/>
            </restriction>
        </simpleType>
</element>
```

// OMG IDL

```
wstring Country;
const short N=5;
wstring Place<N>;
```

7.4.6 Mapping for Any

The ***anyType*** represents an abstraction called the **ur-type** which is the base type from which all simple and complex types are derived. An anyType type does not constrain its content in any way. It is possible to use anyType like other type. It can be mapped to the OMG IDL datatype any.

Here is an example:

```
<!--WSDL -->
<element name="T" type="anyType"/>
```

// OMG IDL

```
any T;
```

7.4.7 Anonymous XML types

Anonymous types are deprecated in CORBA. While it is allowable, in some cases, to map an anonymous XML type

specification to an anonymous IDL type spec (e.g.., for sequences as members of an IDL struct), there are cases which require an explicit IDL type name (e.g., for operation parameters).

Whenever it is explicitly required by the IDL syntax, the anonymous XML types are mapped to an explicit IDL typedef.

The name of the type to use for the generated IDL typedef is constructed by prefixing the name of the element (which has an anonymous XML type specification attached) with the string "T_". In case of collision with another type starting with "T_", the translator will add sufficient extra "_" character(s) to the end of the prefix to resolve the collision.

7.5 Mapping for Complex XML Schema Types

This section shows how complex XML schema types used in WSDL are mapped to CORBA.

7.5.1 Mapping for Sequence Group Element

The sequence element in WSDL specifies that the child elements must appear in the order it is specified. It can be mapped OMG IDL ***struct*** datatype.

Here is an example

```
<!--WSDL -->
<complexType name = “myStruct">
    <sequence>
        <element name="member_1" type="short"/>
        <element name="member_2" type="long"/>
    </sequence>
</complexType>
```

```
// OMG IDL

struct  myStruct  {
    short member_1:
    long   member_2;
};
```

7.5.2 Mapping for Choice Group Element

The choice group element in WSDL allows only one of its children to appear in an instance. It can be mapped to discriminated ***union*** of OMG IDL with the discriminator type taken as IDL datatype ***long***.

Here is an example.

```
<!--WSDL -->
<complexType name="myUnion">
    <choice>
        <element name="c" type="char"/>
        <element name="s" type="short"/>
    </choice>
</complexType>
```

```
// OMG IDL

union myUnion switch (long) {
    case 1: char c;
    case 2: short s;
};
```

7.5.3 Mapping for All Group Element

The all element in WSDL specifies that the child elements do not need to appear in the order they are specified. It is mapped OMG IDL struct datatype, using the same rules as for a Sequence Group Element..

The interaction translator is responsible to arrange the child elements in the proper order to be mapped to the corresponding IDL Struct.

Here is an example

```
<!--WSDL -->
<complexType name = "myAll">
    <all>
        <element name="a_member_1" type="short"/>
        <element name="a_member_2" type="long"/>
    </all>
</complexType>
```

```
// OMG IDL
struct myAll {
    short a_member_1;
    long a_member_2;
};
```

7.5.4 Mapping elements with cardinality constraints to IDL sequence member

For use in the complex type mappings above, there is a special rule for mapping elements of an XML complex types, when those elements have minOccurs=0, or maxOccurs>1.

If an element, which is a member of an XML complex type, has minOccurs=0 or has maxOccurs>1, that element will be mapped to an unnamed IDL Sequence.

Example:

```
<complexType name = "mesgInfoType">
        <xsd:sequence>
                <xsd:element name="infoItem1" type="short"/>
                <xsd:element name="optInfo" type="myStruct"  minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
```

```
// OMG IDL
struct mesgInfoType {
  short infoItem1;
  sequence<myStruct> optInfo;
};
```

7.5.5 Mapping attributes of complex type

Attributes of sequence and all group complex types are mapped as additional members of the IDL struct.

Complex types which use attribute groups have the attributes in that group mapped explicitly as members of the IDL struct, in the same order as if the attribute group definitions were expanded in line.

If a Complex Type with simpleContent has one or more attributes, that complex type is mapped to an IDL Struct, with the first member of the IDL struct being of the simple type and having the name "value". The attributes are each mapped as additional members of the IDL Struct.

If an XML schema attribute is defined anonymously (e.g., it uses an inline enumeration extension of string), the

mapping shall generate an explicit IDL typedef using the "T_" prefix applied to the name of the attribute as the type name, just is for XML schema elements which are defined anonymously.

Attributes of a choice group element are not mapped.

If an attribute of the complex type is optional, then it is mapped to a struct member which is a sequence (just as if it were an element with minOccurs=0). Optional attributes are represented as IDL sequences in order to allow zero members, to cover the case of the attribute not being present.

Example:

```
<xsd:complexType>  taggedShort
        <xsd:simpleContent>
                <xsd:extension base="xsd:short">
                        <xsd:attribute name="type" tag="xsd:string" use="optional"/>
                </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

```
// OMG IDL
struct taggedShort {
  short value;
  sequence<wstring> tag;
};
```

7.6 Mapping for SOAP Array Type

SOAP Array types extends the "***SOAP-ENC:Array***" type defined in the SOAP 1.1 encoding Schema. SOAP array datatype is mapped to either a ***sequence*** or ***array*** OMG IDL construct based on the following cases:

1) One-dimensional SOAP array without size specification maps to OMG IDL unbounded ***sequence*** datatype.

2) One-dimensional SOAP array with size specification maps to OMG IDL bounded ***sequence*** datatype.

3) Multi-dimensional array with size specification maps to OMG IDL ***array*** datatype.

4) Multi-dimensional array without size specification cannot be mapped to an IDL construct and it is valid for the translation mechanism to generate an error for this case.

The name of the mapped IDL sequence or array datatype is generated from the value of the ***<xsd:complexType>*** element.

The type of the SOAP array item, mentioned in the "***type***" attribute of the ***<xsd:element>*** or in the "***wsdl:arrayType***" attribute, becomes the base type of the mapped IDL sequence or array. This type is mapped to an OMG IDL construct according to the datatype mapping rules in this specification.

The IDL ***sequence*** or ***array*** bound is determined from the "***maxOccurs***" attribute of the <***xsd:element***>. If "***maxOccurs***" value is "***unbounded***" for a multi-dimensional array, it can't be mapped.

The dimension of the array is determined from the "***SOAP-ENC:arrayType***" attribute.

The following example shows the mapping of a SOAP Array datatype to OMG IDL unbounded ***sequence***.

```
<!--WSDL -->
<xsd:complexType name = "ArrayOfLong">
```

```
    <xsd:complexContent>
        <xsd:restriction base="SOAP-ENC:Array">
            <xsd:attribute
                ref="SOAP-ENC:arrayType"
                wsdl:arrayType="xsd:int[]"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

// OMG IDL

typedef sequence<long> ArrayOfLong;

The following example shows the mapping of a SOAP ***Array*** datatype to OMG IDL bounded ***sequence***.

```
<!--WSDL -->
<xsd:complexType name = “ArrayOfLong">
    <xsd:complexContent>
        <xsd:restriction base="SOAP-ENC:Array">
            <xsd:sequence>
                <xsd:element
                    name="item" type="xsd:int"
                    minOccurs="10" maxOccurs="10"/>
            </sequence>
            <xsd:attribute
                ref="SOAP-ENC:arrayType"
                wsdl:arrayType="xsd:int[]"/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

// OMG IDL

typedef sequence<long, 10> ArrayOfLong;

7.7 Mapping IDL Name

Normally, names in WSDL map to identical names in IDL. However, names (e.g., IDL keywords) that cannot be used in IDL need to be converted. Following OMG specification “Java™ Language to IDL Mapping Specification”, the conversion below shall be applied.

1) IDL keyword

   If the WSDL identifier clashes with an IDL keyword, prepend an underscore “_” (to form an escaped identifier).

   For example, the WSDL name ***oneway*** is mapped to the OMG IDL identifier ***_oneway***.

2) WSDL Names starting with underscore

   If the WSDL identifier begins with an underscore “_”, the underscore is replaced by “J_”.

   For example, the WSDL name ***_fred*** is mapped to ***J_fred*** in IDL.

3) Unicode characters in WSDL

   IDL does not support Unicode. Thus, if ‘$’ or a kanji character is included in the identifier, it is replaced by “U” and a 4-digit hexadecimal number (in upper case)

   For example, the WSDL name ***a$b*** is mapped to ***aU0024b*** in IDL.

4) Method is overloaded

IDL does not support overloaded methods. If the WSDL operation is overloaded, two underscores "__" are added to the method name, followed by IDL type names of the parameters separated by two underscores "__". A space in the type (like in long long) is replaced with an underscore "_". The underscore at the beginning of an escaped identifier is removed.

For example, if the WSDL mapping results in the following two IDL operations, they are transformed as shown below:

```
void hello();
void hello(in long x, in abc y);
```

This is transformed as below

```
void hello__();
void hello__long__abc(in long x, in abc y);
```

If the in/out parameter names are overloaded in the same method, it is an error. Also, if a method that doesn't include in/out is overloaded, it is an error.

5) WSDL identifiers that differ only in case

IDL names are not case sensitive. Thus, if there are two or more names that are distinguished only by case, an underscore "_" is appended to the original name, and then decimal numbers indicating the positions of the uppercase characters are appended, separated by an underscore. Indices are zero based.

For example, the WSDL names ***jack***, ***Jack***, and ***jAcK*** are mapped to IDL as ***jack_***, ***Jack_0***, and ***jAcK_1_3*** respectively.

However, it is an error for the following names to be distinguished only by case.

- ***module*** name
- ***interface*** name

6) If the identifiers are not unique after application of the mapping rules above, it is an error.

7.8 Identifier Information File

Identifier information file is a text file in XML format that collects identifiers.

- <name>~</name> sets the name before conversion (IDL)
- <name_to>~</name_to> sets the name after conversion (WSDL).
- <name_to>~</name_to> is not generated if identifiers were not converted

.

<module>* <name>~</name> <name_to>~</name_to>? <interface>*	module information interface information
<name>~</name> <name_to>~</name_to>? <typedef>* <name>~</name> <name_to>~</name_to>? </typedef>	 typedef information(includes struct, enum)
<exception>* <name>~</name> <name_to>~</name_to>? </exception>	 exception information
<method>* <name>~</name> <name_to>~</name_to>? </method> </interface> </module>	 method information

*: 0 or more
?: 0 or 1
+: 1 or more

7.9 Input Data

Input data is a WSDL file containing a WSDL document. Structure of a WSDL document is shown below

```
<?xml version="1.0"?>
<definitions    >
     <types>    </types>
     <message>    </message>
     <portType>    </portType>
     <binding>    </binding>
     <service>    </service>
</definitions>
```

7.10 Output Data

7.10.1 IDL File

Format of the generated IDL file is as follows.

```
module module_name {
     interface interface_name {
          typedef  type  type_name;
          exception exception_structure_name {
               data_type member_name;
          };

           type_name  operation_name(in|inout|out
               data_type parameter_name,...)
               [raises( exception_structure_name,.... )];
```

```
        };
    };
```

7.10.2 SOAP Information File

This file contains the information that is missing from IDL but necessary for the SOAP-CORBA gateway. Information described in the SOAP information files is as follows.

Parameter Name	Parameter Value	Description
SOAPAction	Method name and ACTION information	Specify the method name with full scope in the IDL (e.g., **::module1::interface1::op1**). Following the method name, specify the value of ***soapAction*** property in <***soap:body***> element. Separate the method name and the value of ***soapAction*** by one or more space.

7.10.3 Identifier Information File

This file contains a table of original names and converted names, in case names (identifiers) are converted. The <***name***> element is generated even if they are not converted.

A Sample Input and Output of WSDL to IDL

A.1 Input: Sample WSDL

(Example 5: SOAP binding of request-response RPC operation over HTTP. The type of ***startTime*** and ***endTime*** is changed from "***xsd:timeInstant***" to "***xsd:string***".)

```
<?xml version="1.0"?>
<definitions name="StockQuote"
      targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:xsd1="http://example.com/stockquote/schema"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

      <types>
          <schema targetNamespace="http://example.com/stockquote/schema"
              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
              xmlns="http://www.w3.org/2001/XMLSchema">
              <complexType name="TimePeriod">
                  <all>
                      <element name="startTime" type="xsd:string"/>
                      <element name="endTime" type="xsd:string"/>
                  </all>
              </complexType>
              <complexType name="ArrayOfFloat">
                  <complexContent>
                      <restriction base="soapenc:Array">
                          <attribute ref="soapenc:arrayType"
                              wsdl:arrayType="xsd:float[]"/>
                      </restriction>
                  </complexContent>
              </complexType>
          </schema>
      </types>

      <message name="GetTradePricesInput">
          <part name="tickerSymbol" type="xsd:string"/>
          <part name="timePeriod" type="xsd1:TimePeriod"/>
      </message>

      <message name="GetTradePricesOutput">
          <part name="result" type="xsd1:ArrayOfFloat"/>
          <part name="frequency" type="xsd:float"/>
      </message>

      <portType name="StockQuotePortType">
          <operation name="GetTradePrices"
              parameterOrder="tickerSymbol timePeriod frequency">
              <input message="tns:GetTradePricesInput"/>
              <output message="tns:GetTradePricesOutput"/>
          </operation>
      </portType>

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

```
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded"
                    namespace="http://example.com/stockquote"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>

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

A.2 Output: Sample OMG IDL

```
#pragma prefix “http_//example.com"

module stockquote_wsdl {

    interface StockQuotePortType {
        typedef sequence<float> ArrayOfFloat;
        typedef struct TimePeriod {
            wstring startTime;
            wstring endTime;
        };

        ArrayOfFloat GetTradePrices(
            in  wstring tickerSymbol,
            in TimePeriod timePeriod,
            out float frequency);
    };
};
```

A.3 Output: Sample SOAP Information File

```
# SOAP inforamtion file
# list of SOAPAction information
SOAPAction   ::StockQuoteService::StockQuoteSoapBinding::GetTradePrices  http://example.com/GetTradePrices
```

A.4 Output: Sample Identifier Information File

```
<module>
    <name>stockquote_wsdl</name>
    <interface>
        <name>StockQuotePortType</name>
        <typedef>
            <name>ArrayOfFloat</name>
        </typedef>
        <typedef>
            <name>TimePeriod</name>
        </typedef>
        <method>
```

```
            <name>GetTradePrices</name>
        </method>
    </interface>
</module>
```