Document Number: DSP0230

Date: 2009-04-21

Version: 1.0.1

WS-CIM Mapping Specification

Document Type: Specification

Document Status: DMTF Standard

Document Language: E

CONTENTS

Foreword 5
Introduction 6
1 Scope 7
 1.1 In-Scope Features 7
 1.2 Out-of-Scope Considerations 7
2 Conformance 8
3 Normative References 8
 3.1 Approved References 8
 3.2 Other References 9
4 Terms and Definitions 9
5 Symbols and Abbreviated Terms 10
6 Namespace Prefixes and Schema Locations 11
7 Dereferencing Schema URI Locations in Order to Access XML Schema 13
8 Mapping Primitive Datatypes 14
 8.1 cimDateTime Datatype 15
 8.2 CIM References 16
 8.3 cimAnySimpleType Datatype 17
 8.4 Derivation by Restriction 17
 8.5 WS-CIM Canonical Values 17
9 CIM Class to XML Schema Mappings 18
 9.1 Class Namespace 18
 9.2 Properties 18
 9.3 Class Structure 27
 9.4 Class Inheritance 29
 9.5 Method Parameters 30
 9.6 CIM Instances 33
 9.7 Superclass Class Representation 33
10 CIM Methods to WSDL Mappings 35
 10.1 Defining WSDL Message Structures 35
 10.2 Defining WSDL Operation Structures 35
 10.3 Defining wsa:Actions 36
11 Qualifier Mappings 37
 11.1 General Format 37
 11.2 Mapping CIM Qualifiers to XSD Elements 39
 11.3 Inheritance of Qualifiers 41
ANNEX A (Informative) Schemas 43
 A.1 Common WS-CIM Schema: DSP8004 43
 A.2 Qualifiers Schema: DSP8005 46
 A.3 Class Hierarchy Type Schema: DSP8006 48
ANNEX B (Informative) Conventions 49
ANNEX C (Informative) Examples 50
 C.1 MOF Definitions 50
 C.2 XSD 51
 C.3 WSDL Fragments 55
 C.4 MetaData Fragments 56
ANNEX D (Informative) Collation Optimization Available to Implementers 60
 D.1 Sorting with Limited Character Set 60
 D.2 Note of Caution Concerning Collation 60

Tables

Table 1 – Namespaces 11
Table 2 – Namespace Prefixes 12
Table 3 – Schema URI Locations 12
Table 4 – XSD DSP Numbers 12
Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes 14
Table 6 – Rules for Converting datetime to cimDateTime 15
Table 7 – Rules for Converting cimDateTime to datetime 15
Table 8 – CIM Qualifiers Mapped to XSD Elements 39
Table 9 – Rules of Qualifier Inheritance 42

Foreword

The *WS-CIM Mapping Specification* (DSP0230) was prepared by the DMTF WS-Management Working Group.

The authors would like to acknowledge Andrea Westerinen (employed by Cisco at the time and now at Microsoft) for drafting the Charter of the Working Group and initially leading the effort as Chairperson.

Authors:

- Akhil Arora, Sun Microsystems, Inc.
- Ed Boden, IBM
- Mark Carlson, Sun Microsystems, Inc. (past Co-Chair)
- Josh Cohen, Microsoft Corporation
- Asad Faizi, Microsoft Corporation (past Editor)
- Vincent Kowalski, BMC Software, Inc. (past Co-Chair)
- Heather Kreger, IBM
- Richard Landau, Dell Inc.
- Tom Maguire, EMC
- Andreas Maier, IBM
- James Martin, Intel Corporation
- Bryan Murray, Hewlett-Packard
- Brian Reistad, Microsoft Corporation
- Mitsunori Satomi, Hitachi
- Hemal Shah, Broadcom
- Sharon Smith, Intel Corporation
- Kirk Wilson, CA, Inc. (past Editor)
- Dr. Jerry Xie, Intel Corporation
- Steve Hand, Symantec Corporation (Editor and Chair)

Introduction

Management based on the Common Information Model (CIM) in a Web Services environment requires that the CIM Schema (classes, properties, and methods) be rendered in XML Schema and Web Services Description Language (WSDL). To achieve this, CIM must be mapped to WSDL and XML Schema through an explicit algorithm that can be programmed for automatic translation.

This specification provides the normative rules and recommendations that describe the structure of the XML Schema, WSDL fragments, and metadata fragments that correspond to the elements of CIM models, and the representation of CIM instances as XML instance documents. A conformant implementation of a CIM model to XML Schema, WSDL fragments, and metadata fragments transformation algorithm must yield an XML Schema, WSDL fragments, and metadata fragments as described in this specification. These CIM models may be expressed in CIM Managed Object Format (MOF) or in other equivalent ways. Throughout this specification, examples illustrate the mapping from CIM MOF.

WS-CIM Mapping Specification

1 Scope

The goal of this specification is to produce a normative description of a protocol-independent mapping of CIM models to XML Schema, WSDL fragments, and metadata fragments. The features of CIM that are within the scope of this specification correspond to a subset of the features of CIM that are defined in the *CIM Infrastructure Specification*, DSP0004.

Another goal of this specification is to allow the most expedient use of current Web Services (WS) infrastructure as a foundation for implementing a WS-CIM compliant system. This specification has been written to leverage the existing Web Services standards and best practices that are currently widely deployed and supported by Web Services infrastructure. As those standards and best practices evolve, future versions of this specification should evolve to include them.

1.1 In-Scope Features

The following XML Schema, WSDL, and metadata is defined for the Common Information Model (CIM):

- Namespace URIs and the XML Schema definitions for CIM classes and their properties, qualifiers, and methods. The mapping of CIM classes covers regular, association, exception, and indication classes.
- WSDL message definitions for CIM methods. The WSDL mapping supports WSDL version 1.1.
- WSDL portType operation definitions for CIM methods
- Metadata fragments for CIM qualifiers

1.2 Out-of-Scope Considerations

The following items are outside the scope of this specification:

- This specification does not address mapping XML Schema structures to other CIM representations, such as MOF or CIM-XML.
- Features excluded from the scope of this mapping include mapping of CIM instance definitions in MOF and MOF compiler directives (pragmata). (Note that the mapping of CIM instances is addressed in 9.6.)
- A WSDL mapping with portTypes and bindings is not provided by this specification. WSDL bindings are protocol specific.
- Protocol-specific features of CIM or Web-Based Enterprise Management (WBEM), such as CIM Operations over HTTP, the XML Representation of CIM, or WS-Management, are outside the scope of this specification.
- This version of the specification does not provide mappings for Qualifier declarations. This version is limited to the XML Schema definitions of metadata instances (Qualifier values) that correspond to the CIM qualifiers in a CIM model.
- This version does not specify a metadata container for the mapped Qualifier values, but leaves it to the specific protocol to determine where metadata resides.
- This version of the specification does not allow distinguishing empty arrays from arrays that are NULL. This limitation is a result of the decision to use existing standards, which use inline arrays, for representing arrays in XML.

- The invocation of CIM methods may result in an exception represented by one or more instances of classes whose `EXCEPTION` qualifiers are effectively TRUE. While such instances shall be represented in XML according to the mapping rules for CIM classes in clause 9, requirements regarding the transmittal of these exceptions when they occur are protocol-specific and are not in scope for this specification.

2 Conformance

To be compliant with this specification, an XML Schema, WSDL fragment definitions (messages and operations), and metadata elements shall conform to all normative requirements of this specification.

Implementations shall not use the namespaces for CIM classes that conform to this specification (see 9.1) unless the XML Schema for those classes conforms to this specification.

3 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

3.1 Approved References

DMTF DSP0004, *Common Information Model (CIM) Infrastructure Specification*, 2.3
http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

DMTF DSP0226, *Web Services for Management Specification, 1.1.0,*
http://www.dmtf.org/standards/published_documents/DSP0226_1.1.0.pdf

DMTF DSP4009, *Process for publishing XML schema, XML documents and XSLT stylesheets*
http://www.dmtf.org/standards/published_documents/DSP4009_1.0.0.pdf

IETF RFC 3987, *Internationalized Resource Identifiers (IRIs)*, January 2005
http://www.ietf.org/rfc/rfc3987.txt

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005
http://www.ietf.org/rfc/rfc3986.txt

UNICODE COLLATION ALGORITHM, Unicode Technical Standard #10, version 5.1.0, 2008-03-28
http://Unicode.org/reports/tr10

ISO/IEC 10646:2003 *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

W3C, *Namespaces in XML*, W3C Recommendation, 14 January 1999. (This version of the *XML Information Set Recommendation* is available at http://www.w3.org/TR/1999/REC-xml-names-19990114. The latest version of *Namespaces in XML* is available at http://www.w3.org/TR/REC-xml-names.)

W3C, *Web Services Addressing (WS-Addressing) 1.0 – Core*, W3C Recommendation, 9 May 2006,
http://www.w3.org/TR/ws-addr-core/

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

W3C, *XML Schema Part 2: Datatypes*, W3C Recommendation, October 2004
http://www.w3.org/TR/xmlschema-2/

W3C, *XML Schema Part 1: Structures*, W3C Recommendation, October 2004
http://www.w3.org/TR/xmlschema-1/

3.2 Other References

ISO, ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.2
cannot
used for statements of possibility and capability, whether material, physical or causal

3.3
conditional
indicates requirements to be followed strictly in order to conform to the document when the specified conditions are met

3.4
mandatory
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.5
may
indicates a course of action permissible within the limits of the document

3.6
need not
indicates a course of action permissible within the limits of the document

3.7
optional
indicates a course of action permissible within the limits of the document

3.8
shall
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.9
shall not
indicates requirements to be followed strictly in order to conform to the document and from which no deviation is permitted

3.10

should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.11

should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.12

Global Element Declaration

element declaration in an XML Schema that places the element as an immediate child of the root element of the schema

3.13

Managed Object Format

an IDL based language, defined by the DMTF, expressing the structure, behavior, and semantics of a CIM class and its instances

3.14

Uniform Record Identifier

a Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource. See RFC3986.

3.15

Runtime

describes the situation where XML instances are produced that are conformant to this specification

3.16

WS-CIM

of or pertaining to this specification

NOTE: "WS-CIM" is not an acronym; it should be treated simply as the name of the contents of the specification.

3.17

XML instance document

an XML document that conforms to a specified XML Schema

As used in this specification, *XML instance document* refers to a document that conforms to an XML Schema that conforms to the rules in clause 9.

3.18

XSDL

offers facilities for describing the structure and constraining the contents of XML documents, including those which exploit the XML Namespace facility. XSDL documents have the '.xsd' file extension. See: http://www.w3.org/TR/xmlschema11-1/.

5 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

5.1

GED

Global Element Declaration

5.2
MOF
Managed Object Format

5.3
URI
Uniform Resource Identifier

5.4
WSDL
Web Services Description Language

5.5
XML
Extensible Mark-up Language

5.6
XSDL
XML Schema Definition Language

6 Namespace Prefixes and Schema Locations

Table 1 through Table 4 list URIs using the ws-cim-major-version, *X*, and the cim-schema-major-version, *Y*. When using these URIs, replace the *X* and *Y* variables with the actual version numbers.

This specification defines namespaces as shown in Table 1.

Table 1 – Namespaces

Namespace	Description
http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/*ClassName*	Contains the schema for the class *ClassName*
http://schemas.dmtf.org/wbem/wscim/*X*/common	Contains the schema for common elements such as datatypes required for defining XML schemas for CIM classes
http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/qualifiers	Contains the schemas of qualifiers that are mapped to metadata fragments
http://schemas.dmtf.org/wbem/wscim/*X*/classhiertype	Contains the schema definitions for representing the subclass/superclass hierarchy of the CIM Schema as the value of a property
http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/classhierarchy	Contains the GEDs that represent the subclass/superclass hierarchy of the CIM Schema

The namespace prefixes shown in Table 2 are used throughout this document. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see *NameSpaces in XML*).

Table 2 – Namespace Prefixes

Prefix	Namespace
class	http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/*ClassName*
cim	http://schemas.dmtf.org/wbem/wscim/*X*/common
cimQ	http://schemas.dmtf.org/wbem/wscim/X/cim-schema/*Y*/qualifiers
ctype	http://schemas.dmtf.org/wbem/wscim/*X*/classhiertype
chier	http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/classhierarchy
wsa	Any wsa:addressing standard defining EPRs, such as http://www.w3.org/2005/08/addressing (see *Web Services Addressing (WS-Addressing) 1.0 – Core*) or http://schemas.xmlsoap.org/ws/2004/08/addressing (see DSP0226, "Management Addressing" clause*)*
wsdl	http://schemas.xmlsoap.org/wsdl (see *Web Services Description Language (WSDL) 1.1*)
xs	http://www.w3.org/2001/XMLSchema (see *XML Schema Parts 1 & 2*)
xsi	http://www.w3.org/2001/XMLSchema-instance

Table 3 defines the schema location URIs for the schemas defined in this specification.

Table 3 – Schema URI Locations

Prefix	Schema Location URLs
class	http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/*ClassName*.xsd
cim	http://schemas.dmtf.org/wbem/wscim/*X*/common.xsd
cimQ	http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/qualifiers.xsd
ctype	http://schemas.dmtf.org/wbem/wscim/*X*/classhiertype.xsd
chier	http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/classhierarchy.xsd

Table 4 defines the DSP numbers for the XSD files defined by this specification.

Table 4 – XSD DSP Numbers

XSD File Name	DSP Number
http://schemas.dmtf.org/wbem/wscim/*X*/common.xsd	DSP8004
http://schemas.dmtf.org/wbem/wscim/*X*/cim-schema/*Y*/qualifiers.xsd	DSP8005
http://schemas.dmtf.org/wbem/wscim/*X*/classhiertype.xsd	DSP8006

7 Dereferencing Schema URI Locations in Order to Access XML Schema

This clause defines how DMTF is to publish artifacts produced in accordance with this specification.

A client application may construct schema URIs as specified in the following subclause to retrieve the schema documents from the DMTF schema website (http://schemas.dmtf.org/).

DMTF shall publish the XML schema documents listed in Table 3 at the URI locations specified in Table 3. A schema document published at one of these URI locations will always represent the most recent version of the class namespace definition.

DMTF shall also publish productions of CIM classes in XSD schema at locations that support retrieval of specific versions of the class namespace definition as follows:

- At a URI location where the ws-cim-major-version number in the URI specified in Table 3 is replaced with the major, minor, and revision formatted as " major [“.” Minor [“.” Revision]]”of the exact CIM schema version of which the class is a member. All classes published at this URI location shall be final classes.

  EXAMPLE:

  http://schemas.dmtf.org/wbem/wscim/1.1.0/cim-schema/2.17.0/*ClassName*.xsd

- At a URI location where the ws-cim-major-version number in the URI specified in Table 3 is replaced with the major, minor, and revision numbers of the CIM schema version and where a “plus” character (+) is appended to that version number. The format shall be: " major [“.” Minor [“.” Revision]] "+" ". Each such class shall include all experimental content defined for the included version of that class.

  EXAMPLE:

  http://schemas.dmtf.org/wbem/wscim/1.1.0/cim-schema/2.17.0+/*ClassName*.xsd

  EXAMPLE 1: If the latest available final version of the CIM schema is 2.11.0 and the WS-CIM mapping specifications is 1.3.0, the following URI locations would retrieve the same XML Schema file:

  http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/*ClassName.xsd*

  http://schemas.dmtf.org/wbem/wscim/1.3.0/cim-schema/2.11.0/*ClassName*.xsd

  EXAMPLE 2: To retrieve the XML Schema for the same class from CIM schema version 2.10.1 based on WS-CIM mapping version 1.2.0, the following URI location would be used:

  http://schemas.dmtf.org/wbem/wscim/1.2.0/cim-schema/2.10.1/*ClassName*.xsd

  EXAMPLE 3: To retrieve the XML Schema for the same class from CIM schema version 2.11.0 "experimental" based on WS-CIM mapping version 1.3.0, the following URI location could be used:

  http://schemas.dmtf.org/wbem/wscim/1.2.0/cim-schema/2.10.1+/*ClassName*.xsd

7.1.1 Validation of CIM Instances

In some cases, it is desirable to attempt schema validation of the Instance document. However, if the major version of the class or other XML Schemas defined in this specification that are used in the instance document differ from the version of the XML Schemas that the recipient of the instance document has, then validation is impossible. If the major version is the same and the minor version differs, validation is possible.

DMTF permits the structure of the class to change in backwards-compatible ways within a release of a major version of CIM. DSP0004 (see “Schema Versions”) describes the nature of permitted changes in this case. However, the changes permitted could cause XML schema validation errors. Implementations have a choice on how to be resilient to the changes permitted in such cases.

To perform conventional XML schema validation, a validator must obtain the exact schema version used to produce the instance. If the exact class schema document for the received CIM instance is known, it may be retrieved as described previously. The Instance document may indicate the URI location for the Class schema document to which its structure conforms through the xsi:schemaLocation attribute as specified in 9.6. Validation of the CIM instance document with the class schema document retrieved through this mechanism shall be possible.

Alternatively, the recipient may use a custom XML schema validation routine that tolerates the permitted backwards-compatible changes previously referenced.

8 Mapping Primitive Datatypes

Specific WS-CIM datatypes are defined as extensions of simple XSD datatypes. These extended WS-CIM datatypes allow the use of any attribute in conjunction with the simple XSD base datatype that corresponds directly to a CIM datatype. The WS-CIM datatypes are defined in the common.xsd file (ANNEX A).

CIM datatypes are converted to WS-CIM datatypes as shown in Table 5.

Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes

CIM Datatype	Corresponding Base XSD Datatypes	WS-CIM Datatypes
`uint8`	`xs:unsignedByte`	`cim:cimUnsignedByte`
`sint8`	`xs:byte`	`cim:cimByte`
`uint16`	`xs:unsignedShort`	`cim:cimUnsignedShort`
`sint16`	`xs:short`	`cim:cimShort`
`uint32`	`xs:unsignedInt`	`cim:cimUnsignedInt`
`sint32`	`xs:int`	`cim:cimInt`
`uint64`	`xs:unsignedLong`	`cim:cimUnsignedLong`
`sint64`	`xs:long`	`cim:cimLong`
`string`	`xs:string`	`cim:cimString`
`Boolean`	`xs:boolean`	`cim:cimBoolean`
`real32`	`xs:float`	`cim:cimFloat`
`real64`	`xs:double`	`cim:cimDouble`

`datetime`	`xs:duration`
	`xs:date`
	`xs:time`
	`xs:dateTime`
	`xs:string`
	Depending on use-case
	See 8.1.
		`cim:cimDateTime`

`char16`	`xs:string` With maxLength restriction = 1	`cim:cimChar16`
`<class> REF`	`N/A`	`cim:cimReference` See 8.2.

NOTE: For mapping of array properties, see 9.2.2.

CIM properties that are designated with the following qualifiers require special mapping that supersedes the mappings shown in Table 5:

```
Octetstring
EmbeddedInstance
EmbeddedObject
```

See 9.2.4 for mapping of Octetstring properties; see 9.2.5 for mapping of EmbeddedInstance and EmbeddedObject properties.

8.1 cimDateTime Datatype

The `cim:cimDateTime` datatype is defined as follows:

```
<xs:complexType name="cimDateTime">
    <xs:choice>
      <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
      <xs:element name="Interval" type="xs:duration"/>
      <xs:element name="Date" type="xs:date"/>
      <xs:element name="Time" type="xs:time"/>
      <xs:element name="Datetime" type="xs:dateTime"/>
    </xs:choice>
<xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

The rules shown in Table 6 should be used to convert CIM `datetime` to `cim:cimDateTime.`

Table 6 – Rules for Converting datetime to cimDateTime

CIM datetime Use Case	String Condition	cim:cimDateTime Element
interval	String contains ":"	`Interval`
date and time	String contains "+" or "-" and does not contain any asterisks	`Datetime`
time	String contains "+" or "-" and no asterisks in the hhmmss.mmmmmm portion, and only asterisks in the yyyymmdd portion	`Time`
date	String contains "+" or "-" and no asterisks in the yyyymmdd portion, and only asterisks in the hhmmss.mmmmmm portion	`Date`
Other	String asterisks other than indicated above	`CIM_DateTime`

The rules shown in Table 7 should be used to convert `cimDateTime` elements on the client side to their representation in CIM.

Table 7 – Rules for Converting cimDateTime to datetime

cim:cimDateTime Element	Representation of datetime in CIM
`Interval`	CIM `datetime` that is an Interval. Fields that are not significant shall be replaced with asterisks. For example, an interval of 2 days 23 hours would be converted to 0000000223****.******:000
`Datetime`	CIM `datetime` that is a time stamp. The mapping of timezone offset is left to the implementer to either normalize it to zero or preserve it in translation. Note that the resulting CIM `datetime` string does not contain any asterisks, because this XML element is used only when the original CIM `datetime` satisfies this condition.
`Time`	CIM `datetime` that is a time stamp. The mapping of timezone offset is left to the implementer to either normalize it to zero or preserve it in translation. Note that the resulting CIM `datetime`

cim:cimDateTime Element	Representation of datetime in CIM
	string does not contain any asterisks in the hhmmss.mmmmmm portion, and contains only asterisks in the yyyymmdd portion, because this XML element is used only when the original CIM `datetime` satisfies this condition.
`Date`	CIM `datetime` that is a time stamp. The mapping of timezone offset is left to the implementer to either normalize it to zero or preserve it in translation. Note that the resulting CIM `datetime` string does not contain any asterisks in the yyyymmdd portion, and contains only asterisks in the hhmmss.mmmmmm portion, because this XML element is used only when the original CIM `datetime` satisfies this condition.
`CIM_DateTime`	CIM `datetime` with a string equal to the XML element text.

8.2 CIM References

The `cim:cimReference` datatype is defined as follows:

```
<xs:complexType name="cimReference">
  <xs:sequence>
    <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
<xs:anyAttribute namespace="##any" processContents="lax"/>

</xs:complexType>
```

The `xs:any` element in this definition represents a structure of a single transport reference that uniquely identifies a location to which messages may be directed for the referenced entity. This structure may be either a single element that expresses the complete transport reference or a sequence of elements, if the transport reference requires multiple elements to uniquely identify a location. In the case of Addressing (see *Web Services Addressing (WS-Addressing) 1.0 – Core* and DSP0226, "Management Addressing" clause), the `xs:any` element shall be replaced by the required wsa:EndpointReference child elements defined by Addressing recommendations, as if the property element were of type wsa:EndpointReferenceType. These requirements for the representation of the reference datatype supersede any requirements specified in DSP0004 regarding the syntactical representation of a value of type reference.

The attribute `maxOccurs="unbounded"` shall not be misconstrued as allowing multiple transport references.

EXAMPLE: An example of the use of Addressing versions as a transport reference, mapped to the `AssociatedComponent` property, is as follows:

```
<xs:element name="AssociatedComponent" type="cim:cimReference"/>
```

The reference could appear in an XML instance document as in either of the following examples:

```
<AssociatedComponent

  xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>. . .</wsa:Address>
 . . .  <!--  Other EPR elements as defined in the 2005/08 specification  -->
</AssociatedComponent>

```

```
<AssociatedComponent

  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">

      <wsa:Address>. . .</wsa:Address>

. . .  <!--  Other EPR elements as defined in the 2004/08 specification   -->

</AssociatedComponent>
```

8.3 cimAnySimpleType Datatype

WS-CIM also introduces a special type built on `xs:anySimpleType`, `cimAnySimpleType`. `cimAnySimpleType` extends `xs:anySimpleType` with the facility to add any attribute to an instance of this type in an XML instance document. This special datatype is required in the mapping of CIM properties with ValueMaps containing ranges because of a restriction in the XML Schema specification. CIM properties with ValueMaps containing ranges are mapped as restrictions of a WS-CIM datatype where the restriction contains an `xs:union` consisting of an explicit enumeration of any discrete values (if any) and the specific ranges specified by the ValueMap (see 9.2.3 for the mapping rules for properties with ValueMaps). However, XML Schema currently requires that the content of a restriction be the same as or be derived from the content type of the parent complex type that is being restricted. Because an XSD union can be of any XSD simple type, XML Schema restricts the use of a union in a derivation by restriction to a parent type whose content is of any simple type. Thus, CIM properties with ValueMaps containing ranges are mapped to XSD elements of the `cimAnySimpleType` datatype.

However, this mapping overrides the normal datatype mapping of the CIM property (as mapped in Table 5). Using the `cimAnySimpleType` datatype means that standard WS-CIM datatyping information is lost for the property. Consequently, the following normative rule governs the use of this special datatype:

The `cimAnySimpleType` datatype shall be used only for mapping CIM properties with ValueMaps containing ranges. Any other use of this datatype is considered non-conformant to the WS-CIM specification.

8.4 Derivation by Restriction

The purpose of the WS-CIM datatypes is to provide the ability to add any attributes to CIM data in the instance document where those attributes are not defined in the XSD definition of the data. For example:

```
. . .
<this:Name xns:AdditionalAttribute=". . . ">
    myName
</this:Name>
. . .
```

where AdditionalAttribute is a global attribute defined in a namespace (xns) and is not explicitly specified in the type definition of Name. Including the AdditionalAttribute attribute in the instance document is valid based on the presence of the following wildcard specification in the definition of the type for this data:

```
  <xs:anyAttribute namespace="##any" processContents="lax"/>
```

However, the anyAttribute wildcard is not inherited by a type definition that is derived by restriction from a parent type containing the wildcard. Therefore, the following normative rule applies to all derivations by restriction:

> To preserve attribute extensibility, the anyAttribute wildcard shall be specified in any derivation by restriction from a WS-CIM datatype.

NOTE: All mapping rules involving a derivation by restriction in this specification explicitly stipulate the inclusion of the anyAttribute wildcard in the mapping.

8.5 WS-CIM Canonical Values

The WS-CIM specification maps a CIM Boolean to an XSD Boolean. According to XSD data types (http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#boolean), a Boolean value can be one of four possible values: true, false, 1, or 0.

To promote interoperability, the WS-CIM specification requires adoption of canonical representation by W3C XPath spec for XSD Boolean type. The implementations shall use the values of TRUE and FALSE as the canonical values for Boolean type.

9 CIM Class to XML Schema Mappings

This clause contains the normative rules for mapping CIM elements into structures of XML Schema. Each clause provides the following information:

- complete normative rules
- an example of the use of those rules
- runtime normative rules and examples, if necessary, to address runtime consideration

9.1 Class Namespace

Each CIM class has an assigned XML namespace, the *class namespace*. This clause defines the class namespace, and subsequent clauses define how the class namespace is used in the mapping.

The rules for specifying the XSD namespace of a CIM class are as follows:

- Each CIM class shall be assigned its own namespace in the XML schema.

  'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-major-version '/' cim-class-schema '_' cim-class-name

  Where:

    wscim-major-version is the major version number of this specification. Note that this version number changes only if there are incompatible changes in the specification.

    cim-schema-major-version is the major version number of the CIM schema version to which the class being converted belongs. Note that this version number changes only if there are incompatible changes in the CIM schema.

    cim-class-schema is the CIM schema name of the class (for example, "CIM"). Note that the schema name may be vendor specific in the case of vendor extensions to CIM classes.

    cim-class-name is the name of the CIM class.

- The process and rules for the publication of the schema documents that define class namespaces are defined in DSP4009.

  EXAMPLE: The `CIM_ComputerSystem` class that belongs to version 2.11.0 of the CIM schema would have the following namespace:

    http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ComputerSystem

9.2 Properties

This clause describes the general principles for converting CIM properties to XSD. It also describes specific principles for converting array properties, value maps, octetstrings, and embedded objects and instances.

9.2.1 General Principles

This clause defines and illustrates the normative rules that apply to the mapping of all CIM properties to XSD.

9.2.1.1 General Rules

The rules for mapping CIM properties to XML Schema are as follows:

- Every property of a CIM class shall be represented by a global element definition. Note that this rule applies to properties locally defined on the class itself and properties inherited from superclasses (see 9.4). The GED that corresponds to a CIM property shall exhibit the following features:
  - The GED shall be defined in the namespace of the class in the XML Schema (see 9.1).
  - The name of the GED shall be the same as the name of the CIM property.
  - The type of the GED shall comply with the datatype conversion table defined in clause 8. However, in some cases, depending on what qualifiers apply to the CIM property, it is necessary to restrict the default type of the GED element. The complete specification of the type of the GED shall comply with the normative rules for mapping qualifiers. (See 11.2 for the normative rules for mapping specific qualifiers to XSD structures.)
- The GED of properties that are not arrays shall be specified with `nillable="true"`, if the CIM property has a `Key` or `Required` qualifier with an effective value of `false`. The GED of properties that are not arrays shall be specified without the `nillable` attribute (the default of `nillable` is `false`), if the CIM property has a `Key` or `Required` qualifier with an effective value of `true`. (See 11.3 for rules regarding the inheritance of qualifiers.)

  NOTE: These rules do not apply to array properties. All GEDs that represent array properties shall be specified with `nillable="true"`.
- The CIM Schema may assign default initializer values directly to properties, as, for example, in the MOF construct `uint16 EnabledState=3` (see DSP0004). Default initializer values shall be mapped to a metadata fragment using `<cim:DefaultValue>`. Default initializer values shall not be mapped to the `xs:default` attribute, which carries different semantics than CIM Schema default values.

  The metadata fragment shall contain the `xsi:type` attribute, which specifies the primitive datatype of the default initializer value. The specified datatype shall be the same as the base type of the WS-CIM defined datatype of the XSD element that represents the CIM property (see clause 8). The base type of a WS-CIM defined datatype shall be determined from its datatype definition in the common namespace. In the case of the `cimDateType` datatype, `xsi:type` shall specify the primitive datatype of the element that is used to express the value of `cimDateType`.

  NOTE: This rule precludes specifying default initializer values for REF properties (`cimReference` elements in XSD mapping). However, specifying such default initializer values is also unsupported in CIM. Values for `cimReference` elements can be provided only at runtime.

The GEDs that represent CIM properties are referenced by child elements within the element to which the CIM class that owns the properties is mapped (see 9.3). Rules for specifying the `minOccurs` and `maxOccurs` attributes of the elements that reference the GEDs are provided in 9.3.1.

EXAMPLE: As an example of the preceding rules, consider the following MOF fragment that defines the (hypothetical) class `EX_BaseComponent`. The complete definition of this class is presented in C.1.1.

```
class EX_BaseComponent {
   datetime InstallDate;
     [...
        Required, MaxLen ( 1024 ) ]
   string Name;
   string StatusDescriptions[];
   string HealthStatus;
};
```

Four GEDs need to be generated to represent the four properties of this class. Based on the preceding rules, the first two properties are mapped as follows:

```
<xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
<xs:element name="Name">
  <xs:complexType>
    <xs:simpleContent>
     <xs:restriction base="cim:cimString">
       <xs:maxLength value="1024"/>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
     </xs:restriction>
    </xs:simpleContent>
  </xs:complexType
</xs:element>
```

The complete mapping of this class is presented in C.2.1. For an example of a default initializer value metadata fragment, see C.4.2.

9.2.1.2 Runtime Rules for Attribute Value Assignment

The occurrence of properties in an XML instance document may be subject to the following rule:

- If a `Key` qualifier that has an effective value of `true` is associated with the CIM property, the `cim:Key` attribute may be applied to the corresponding element in the XML instance document. Use of the `cim:Key` attribute shall conform to the following rules:
  - If the attribute is present, its value shall be assigned as `true` in the XML instance.
  - If the application decides to apply the `cim:Key` attribute to the property, it shall apply it to all properties in the class that have a `Key` qualifier with an effective value of `true` associated with them. If a `Key` qualifier is not associated with the CIM property, this attribute shall be omitted.

The `Name` property can be designated with a `Key` qualifier in CIM:

```
class EX_SomeClass {
  ...
 [... Key]
  string Name;
}
```

The instance document may specify the `cim:Key` attribute for this property as follows:

```
<EX_SomeClass>
  ...
   <Name cim:Key="true">MyName</Name>
  ...
</EX_SomeClass>
```

The following clauses discuss the mapping of more complex CIM properties.

9.2.2 Array Properties

This clause defines and illustrates the specific rules for mapping CIM properties that are arrays.

9.2.2.1 General Rules

The rule for representing arrays is as follows:

Mapping of array properties shall follow the general principles for mapping properties in 9.2.1.1.

NOTE 1: Array properties have a multiplicity (`minOccurs` and `maxOccurs`) that corresponds to the specification of their size in CIM. Rules for specifying the `minOccurs` and `maxOccurs` attributes to the element that represents an array property are provided in 9.3.1.

NOTE 2: Inline arrays represent the current best practices and standards for mapping arrays to XML. New work is beginning to explore alternative array-mapping strategies, and the committee shall track the progress of those efforts for possible inclusion in future versions of this specification.

EXAMPLE 1: Consider the array `StatusDescriptions` in the preceding MOF class definition, which is defined as follows:

```
[...
   ArrayType ( "Indexed" ) ]
string StatusDescriptions[];
```

EXAMPLE 2: This array is defined as an element of the following complex type:

```
<xs:element name="StatusDescriptions" type="cim:cimString" nillable="true"/>
```

EXAMPLE 3: This array property, consisting of the following entries, appears in an XML instance document as follows:

```
<EX_BaseComponent>
  ...
  <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
  <StatusDescriptions>AnotherStatusDescription</StatusDescriptions>
  <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
  ...
</EX_BaseComponent>
```

EXAMPLE 4: The MOF may also specify qualifiers that apply to each element in the array (for example, the maximum length of each element).

NOTE 3: This example is not illustrated in the example class used in this specification.

```
[...
  MaxLen ( 64 ) ]
string StatusDescriptions[];
```

EXAMPLE 5: In the following example, this restriction is defined on the `StatusDescriptions` element using an anonymous complex type definition. The restriction base is the datatype that would otherwise have been assigned to the element itself. See 11.2 for more information about applying qualifiers as restrictions.

```
<xs:element name="StatusDescriptions nillable="true">
  <xs:complexType>
    <xs:restriction base="cim:cimString">
       <xs:maxLength value="64"/>
       <xs:anyAttribute namespace="##any" processContents="lax"/>
     </xs:restriction>
  </xs:complexType>
</xs:element>
```

9.2.2.2 Runtime Rules for Arrays

Specific rules for representing arrays in XML instance documents may apply at runtime:

- The position of each member of an array in its XML representation shall conform to semantics regarding index and value defined by the `ArrayType` qualifier in the *CIM Infrastructure Specification*, DSP0004. Array index is inferred by the position of an element relative to peer elements of the same name.
- Indexed arrays that include members that have a NULL value shall include each such member in the XML representation of the array as an empty element with the `xsi:nil` attribute for these elements set to the value `true`.

The `StatusDescriptions` array is an indexed array. If the second entry were deleted from the array, the preceding example must be transmitted as follows:

```
<EX_BaseComponent>
  ...
  <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
  <StatusDescriptions xsi:nil="true"/>
  <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
  . . .
</EX_BaseComponent>
```

9.2.3 Properties with a ValueMap Qualifier

This clause defines and illustrates the rules for mapping CIM properties with a `ValueMap` qualifier to XSD.

The `ValueMap` qualifier shall be mapped as metadata fragments (see 11.1.2). The `ValueMap` qualifier shall also be mapped in the XSD, according to the following rules.

Mapping of properties qualified with a `ValueMap` qualifier shall follow the general principles for mapping properties in 9.2.1.1, with the following additions:

- If the ValueMap consists of only discrete values, the ValueMap shall be mapped to a `<xs:restriction>` consisting of an enumeration as follows:
  – The base type of the restriction shall be the WS-CIM datatype corresponding to the CIM datatype of the property (see 8).
  – Each discrete value of the ValueMap shall be mapped to a corresponding `<xs:enumeration>` element within the restriction.
- If the ValueMap contains a value specifying a range, the whole ValueMap shall be mapped to a `<xs:restriction>` consisting of a `<xs:union>` as follows:
  – The base type of the restriction shall be `cim:cimAnySimpleType` (see 8.3).
  – The elements of the `<xs:union>` shall be determined according to the following rules:
    - Discrete values shall be mapped to elements to an `<xs:restriction>` as described in the first rule with the exception that the base type of the restriction shall be the corresponding base XSD type of the CIM datatype of the property (see clause 8);
    - Bounded ranges (*m..n*) shall be mapped to an `<xs:restriction>` consisting of `<xs:minInclusive="`*m*`">` and `<xs:maxInclusive="`*n*`">` where the restriction base type shall be the corresponding base XSD type of the CIM datatype of the property;
    - Unbounded ranges open on the left (*…n*) shall be mapped to an `<xs:restriction>` consisting of `<xs:maxInclusive="`*n*`">` where the restriction base type shall be the corresponding base XSD type of the CIM datatype of the property;
    - Unbounded ranges open on the right (*m…*) shall be mapped to an `<xs:restriction>` consisting of `<xs:minInclusive="`*m*`">` where the restriction base type shall be the corresponding base XSD type of the CIM datatype of the property;
    - Open ranges (..) shall be mapped to an `<xs:union>` consisting of all discrete values and/or ranges that are unclaimed by the other values and ranges in the ValueMap by applying the preceding rules for constructing the elements of the `<xs:union>` recursively.

EXAMPLE 1: The following MOF fragment contains only discrete values for `ValueMap`:

```
 [...
  ValueMap { "OK", "Error", "Unknown" } ]
string HealthStatus;
```

The `HealthStatus` property is therefore mapped as follows:

```
<xs:element name="HealthStatus" nillable="true">
    <xs:complexType>
      <xs:simpleContent>
        <xs:restriction base="cim:cimString">
          <xs:enumeration value="OK"/>
          <xs:enumeration value="Error"/>
          <xs:enumeration value="Unknown"/>
          <xs:anyAttribute namespace="##any" processContents="lax"/>
        </xs:restriction>
      </xs:simpleContent
  </xs:complexType>
</xs:element>
```

EXAMPLE 2: The following MOF fragment contains discrete values and bounded ranges for `ValueMap`:

```
 [...
ValueMap { "0", "1", "2", "3..15999", "16000..65535" },
       Values { "Unknown", "Other", "Not Applicable", "DMTF Reserved",
          "Vendor Reserved" }]
   uint16 PortType;
```

The `PortType` property is therefore mapped as follows:

```
<xs:element name="PortType" nillable="true">
    <xs:complexType>
       <xs:simpleContent>
           <xs:restriction base="cim:cimAnySimpleType">
                 <xs:simpleType>
                    <xs:union>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedShort">
                                <xs:enumeration value="0"/>
                                <xs:enumeration value="1"/>
                                <xs:enumeration value="2"/>
                            </xs:restriction>
                        </xs:simpleType>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedShort">
                                <xs:minInclusive value="3"/>
                                <xs:maxInclusive value="15999"/>
                            </xs:restriction>
                        </xs:simpleType>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedShort">
                                <xs:minInclusive value="16000"/>
                                <xs:maxInclusive value="65535"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
```

```
            </xs:restriction>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

EXAMPLE 3: The following MOF fragment contains discrete values, an open range, and an unbounded range for the `ValueMap`:

```
 [...
ValueMap { "1", "2", "3", "4", "5", "6", "7", "..", "16000.." },
       Values { "Other", "Create", "Delete", "Detect", "Read", "Write",
          "Execute", "DMTF Reserved", "Vendor Reserved" }]
   uint16 Activities;
```

The `Activities` property is therefore mapped as follows:

```
<xs:element name="Activities" nillable="true">
    <xs:complexType>
        <xs:simpleContent>
           <xs:restriction base="cim:cimAnySimpleType">
                 <xs:simpleType>
                     <xs:union>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedShort">
                                <xs:enumeration value="1"/>
                                <xs:enumeration value="2"/>
                                <xs:enumeration value="3"/>
                                <xs:enumeration value="4"/>
                                <xs:enumeration value="5"/>
                                <xs:enumeration value="6"/>
                                <xs:enumeration value="7"/>
                            </xs:restriction>
                        </xs:simpleType>
                        <xs:simpleType>
                          <xs:union>
                             <xs:simpleType>
                                <xs:restriction base="xs:unsignedShort">
                                  <xs:enumeration value="0"/>
                                </xs:restriction>
                             </xs:simpleType>
                             <xs:simpleType>
                               <xs:restriction base="xs:unsignedShort">
                                 <xs:minInclusive value="8"/>
                                 <xs:maxInclusive value="15999"/>
                               </xs:restriction>
                             </xs:simpleType>
                          </xs:union>
                        </xs:simpleType>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedShort">
                                <xs:minInclusive value="16000"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
            </xs:restriction>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

9.2.4 Octetstring Properties

The `Octetstring` qualifier may be applied to either `uint8` arrays or `string` arrays. In `uint8` arrays, the property identifies only a single binary entity; in `string` arrays, each string in the array represents a different binary entity.

9.2.4.1 General Rules

The rules for representing properties that are octetstrings are as follows:

- A `uint8` array that is designated as an octetstring shall be mapped to a single XSD element of the type `cim:cimBase64Binary`. The rules for mapping properties defined in 9.2.1.1 apply to this mapping.
- A `string` array that is designated as an octetstring shall be mapped to an array of type `cim:cimHexBinary`. The rules for mapping arrays defined in 9.2.2.1 apply to this mapping.

EXAMPLE 1: The following uint8 array is designated as an octetstring:

```
[...
  Description ("The DER-encoded raw public key. " ),
  OctetString ]
uint8 PublicKey[];
```

It would be represented by the following XSD:

```
<xs:element name="PublicKey" type="cim:cimBase64Binary" nillable="true"/>
```

It would be represented in an XML instance document by entries such as the following:

```
<PublicKey>AAAAExEiM0RVZneImaq7zN3u/w==</PublicKey>
```

EXAMPLE 2: The following CIM `string` array is designated as an octetstring:

```
[…
  Description ("A CRL, or CertificateRevocationList, is a list of certificates which the "
       "CertificateAuthority has revoked and which are not yet expired. " ),
  Octetstring]
string CRL[];
```

It would be represented by the following XSD:

```
<xs:element name="CRL" type="cim:cimHexBinary" nillable="true"/>
```

It would be represented in an XML instance document by entries such as the following:

```
<CRL>0x000000F976H8A4...</CRL>
<CRL>0x000000C675D4G1...</CRL>
<CRL>0x000000D8B1H335...</CRL
```

9.2.4.2 Runtime Value Conversion Rules

This clause defines the normative rules for the runtime conversion rules for values of octetstring properties.

The hex format for the `string` array variant of octetstrings is used to avoid additional conversion steps in the XML protocol layer, which would need to convert the hex encoding generated by the CIM provider to binary and then convert that binary to base64. The values in the preceding examples (see 9.2.4.1) are obtained by applying the following runtime value conversion rules:

- A `uint8` array that is designated as an octetstring shall be converted to its corresponding representation in base64Binary such that the ordered set of array elements is concatenated into a binary multi-octet string, which is converted to base64 encoding. This encoding represents the base64Binary value. The order of the unsigned 8-bit integer array shall be preserved when mapped to the characters of the XML value.

- A `string` array that is designated as an octetstring shall be converted to its corresponding representation in hexBinary such that for each string array element, one hexBinary element is created, with the unchanged value of the string array element.

NOTE: The four-octet length, which constitutes the first four octets of each CIM octetstring, shall be preserved as part of the binary encoding.

9.2.5 EmbeddedObject and EmbeddedInstance Properties

`EmbeddedObject` and `EmbeddedInstance` qualifiers apply to string properties whose values are complete encodings of the data of an instance or class definition. An EmbeddedObject property may contain either the encoding of an instance's data or a class definition; an EmbeddedInstance property contains only the encoding of an instance's data.

9.2.5.1 General Rules

The rule for represented string properties that are designated as EmbeddedObjects or EmbeddedInstances is as follows:

The general rules for mapping properties in 9.2.1.1 apply to properties that contain embedded objects or instances, with the following exception: The property shall be converted to an element of type `xs:anyType`.

EXAMPLE: The following MOF fragment defines a string property that contains an embedded object:

```
[...
  EmbeddedObject ( ″...″ ) ]
string TheObject;
```

It would have the following XSD representation:

```
<xs:element name="TheObject" type="xs:anyType" nillable="true"/>
```

9.2.5.2 Runtime Value Conversion Rules

Runtime conversion of actual values of an EmbeddedInstance or EmbeddedObject property requires different algorithms depending on the representation in the property. For example, the encoding of the instance or class may be provided through MOF or CIM-XML encoding.

This clause defines the normative rules for the runtime conversion of embedded instances and embedded objects, as follows:

- If the CIM property that is qualified by an `EmbeddedInstance` or an `EmbeddedObject` qualifier contains an instance, then
  - – The property value shall be converted to an XML instance representation as if the XSD type of the property was the actual XSD type of the class of the instance.
  - – The property element shall contain an `xsi:type` attribute with the XSD type of the class of the instance (see 9.3.1).
- If the CIM property that is qualified by an `EmbeddedObject` qualifier contains a class definition, the property value shall be converted to the XML Schema of that class. See 9.6 for the normative rules for representing CIM instances.

EXAMPLE: The following class definition in MOF embeds an instance of CIM_Part in CIM_Component:

```
class CIM_Part {
   string Label;
   int PartNo;
};
class CIM_Component {
   [Key]
   string ID;
   [EmbeddedInstance("CIM_Part")]
   string Part;
};
```

Given an embedded instance of CIM_Part with `Label="Front Panel"` and `PartNo="19932"`, the following is a valid instance representation in the runtime XML instance document:

```
<CIM_Component xmlns="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Component"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
   <ID>ua123</ID>
   <Part xmlns:e="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Part"
     xsi:type="e:CIM_Part_Type">
       <e:Label>Front Panel</e:Label>
       <e:PartNo>19932</e:PartNo>
   </Part>
</CIM_Component>
```

9.3 Class Structure

This clause describes the XSD representation of CIM classes. The intended scope is all classes defined in CIM, including associations, indications, and exceptions. Associations, indications, and exceptions are distinguished by having an effective `Association`, `Indication` or `Exception` qualifier, respectively.

NOTE: These qualifiers have standard mappings to metadata fragments for these classes (see 11.1.1).

9.3.1 General Rules

CIM classes are represented in the XML Schema according to the following rules:

- The structure of the CIM class shall be mapped to an XML global complex type definition. The definition of this complex type shall comply with the following rules:
  - – The name of this type shall match that of the CIM class name, including the CIM schema name, with the suffix `_Type`.
  - – The complex type shall consist of an `<xs:sequence>` that contains the set of elements referring to the GEDs that define the properties of the class (see 9.2). These elements have the following form:

    ```
    ‘<xs:element ref=’ QName ‘. ‘ Attributes ‘/>’
    ```

    Where:
    - QName is the QName of the GED that represents the target property.
    - Attributes represents any required attributes (such as `minOccurs` and `maxOccurs`).

  Elements that belong to the class complex type should be ordered by Unicode code point (binary) order of their CIM property name identifiers.

  - In existing service implementations, the collation sequence of property name identifiers should be in Unicode code point (binary) order.
  - In existing service implementations, the collation sequence of property name identifiers may be according to the Unicode Collation Algorithm (UCA) with its default settings. This algorithm is deprecated and should not be used in future service implementations.
  - In DMTF XML schema representations of CIM classes, and in new service implementations, the collation sequence of property name identifiers shall be in code point (binary) order.

The following rules apply to specifying the multiplicity of these elements:

- All elements that do not represent array properties shall have `minOccurs="0"` except for elements that correspond to properties that are designated with a `Key` or `Required` qualifier with an effective value of `true`. Elements that do not represent arrays and represent key and required properties shall have `minOccurs="1"`. Because 1 is the default value for `minOccurs`, it does not need to be explicitly expressed.
- All elements that represent array properties shall have `minOccurs="0"`. If the array size is specified in the CIM definition, the array property shall have `maxOccurs="array size"`. If the array size is not specified, the array property shall have `maxOccurs="unbounded"`.
- All elements except arrays shall have `maxOccurs="1"`. Because 1 is the default value for `maxOccurs`, it does not need to be explicitly expressed.
- Array properties (see 9.2.1.1) shall have a multiplicity that corresponds to the specification of their size in CIM. A bounded array in CIM shall be specified with a `maxOccurs` equal to the size of the array. If no size is specified in CIM, the schema element shall be specified with `minOccurs="0"` and `maxOccurs="unbounded"`.
- The schema of the CIM class shall support an open schema. Open schema means different protocols are able to add protocol-specific elements to instance documents.
  - To allow Open Content, the final element in the sequence shall be as follows:

    ```
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
       maxOccurs="unbounded"/>
    ```

  - To allow attributes to be added to the element that represents the CIM class, following the sequence, the complex type shall allow any attribute to be added to the class with an `xs:anyAttribute` element, as follows:

    ```
    <xs:anyAttribute namespace="##any" processContent="lax"/>
    ```

- The class itself shall be represented by a GED of the type defined in the preceding rule. The name of this element shall be the name of the CIM class including its CIM schema name.

EXAMPLE: The class defined in 9.2.1 has the following mapping as an XSD class definition:

```
<xs:complexType name="EX_BaseComponent_Type">
  <xs:sequence>
    <xs:element ref="class:HealthStatus" minOccurs="0"/>
    <xs:element ref="class:InstallDate" minOccurs="0"/>
    <xs:element ref="class:Name"/>
    <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
    <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContent="lax"/>
</xs:complexType>
<xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type"/>
```

The complete mapping of this class is provided in C.2.1.

9.3.2 Runtime Property Value Rules

Runtime inclusion of property values for instance documents based on the XML Schema for CIM classes is defined by the following rules:

- For "get" operations, CIM service implementations returning the class's GEDs may omit schema-optional (`minOccurs="0"`) properties that have NULL values from responses. Clients are to interpret such omitted properties as having NULL values for those properties.
- Empty arrays (arrays with no members) shall not be included in responses to "get" requests. If the array is required, clients shall interpret the absence of all elements for the array to mean that the array is empty (no members). If the array is not required, clients shall interpret the absence of all elements for the array to mean that the array is either empty or NULL.
- A WS-CIM server shall return all current elements of an array.
- For "set" operations using the class's GEDs, clients may omit schema-optional (`minOccurs="0"`) properties. Service interpretation of the absence of such properties is protocol dependent.
- An instance document conformant to this specification shall include any elements from a foreign namespace at the end of the sorted list of CIM class elements. Elements from a foreign namespace may appear in any order.
- If a `Version` qualifier is associated with the CIM class, the `cim:Version` attribute may be applied to the element that represents the class in an XML instance document. Use of the `cim:Version` attribute shall conform to the following rule:

  If the attribute is present, its value shall be assigned as the value of the `Version` qualifier that is associated with the CIM class, in the XML instance. If the CIM class does not have the `Version` qualifier associated with it, this attribute shall be omitted.

The MOF typically contains the `Version` qualifier, which specifies the latest version of the CIM class in CIM:

```
  [... Version ( "2.10.2") ]
class EX_SomeClass {
...
}
```

The class may be represented in an instance document as follows:

```
<EX_SomeClass cim:Version="2.10.2">
  …
</EX_SomeClass>
```

9.4 Class Inheritance

CIM inheritance is not modeled in the XML Schema of classes or within XML instances.

Class inheritance is governed by the following rules:

- Besides including the GEDs for the properties defined in a class (see 9.2.1.1), the namespace for a class shall also include the GEDs for properties inherited from its superclasses. The class type definition shall contain references to GEDs for all properties defined in and inherited into the class according the rules in 9.3.1.
- In general, classes inherit all properties specified in or inherited by their superclasses along with all qualifiers that are specified as `ToSubClass`. However, properties with the same name may be encountered within an inheritance chain. The `Override` qualifier determines special behaviors that shall be observed by conversion algorithms when encountering properties with duplicate names in the inheritance chain. The following rules govern the mapping of properties with duplicate names:

  - When multiple properties in the inheritance chain that have the same name are not overridden (that is, the effective value of the `Override` qualifier throughout the inheritance chain is `NULL`), the property and its qualifiers in the most derived class shall be mapped. All other duplicate named properties shall not be mapped.
  - When a property in a derived class overrides another property (of the same name and type) in a superclass, the property in the most derived class shall be mapped, including all qualifiers inherited from the overridden property. The overridden property shall not be mapped.
- The inheritance of qualifiers that pertain to properties shall comply with the inheritance rules regarding qualifiers in C.3.
- The definition of a derived class shall follow all other rules for constructing classes as defined in 9.3.1.

NOTE: The metadata fragments for a property shall include any inherited qualifiers, subject to the qualifier inheritance rules defined in 11.3. For more information about metadata fragments, see clause 11.

Inheritance rests on the same type of examples presented in 9.2 and 9.3. The only addition is that the properties inherited from a class's superclasses are included in the GEDs and class complex type definition. For a complete example of inheritance, see C.2.2.

9.5 Method Parameters

The invocation of a CIM extrinsic method is represented by two separate messages:

- the request input message, which represents the invocation of the method and the set of input parameters
- the response output message, which represents the output parameters and the method return value in the successful case

This clause specifies the XML Schema for these elements. These elements are then included as parts in the WSDL input and output messages (see 10.1).

9.5.1 General Rules

The rules in this clause apply to mapping method input and output parameters and method return values.

The GED used for the request input message is called the *input message GED*. The GED used for the response output message is called the *output message GED*. The following rules specify the definition of these GEDs:

- The class namespace of the CIM class being mapped shall contain the input and output message GEDs of all methods owned by the class and inherited from the superclasses. See 9.5.2, which defines class ownership of methods inherited from superclasses.
- The names of these GEDs are defined by the following rules:
  - The `name` of the input message GED shall be the name of the CIM method with `_INPUT` appended.
  - The `name` of the output message GED shall be the name of the CIM method with `_OUTPUT` appended.
- Each GED shall be defined as a complex type containing an `xs:sequence` of in-line elements that represent the respective input or output parameters and return value of the CIM method as immediate children. This structure is further defined in the rest of this clause.

The following rules define the mapping of individual input and output parameters and the return value of the CIM method, and the structure of the complex type used for defining the GEDs:

- Input and output parameters of a CIM method shall be mapped to elements with the same name as the corresponding parameter name. The following rules define the features of these elements:
  - The type of an element that represents an input or output parameter shall be mapped as defined in clause 8.
  - Parameters that are not qualified with a `Required` qualifier shall be mapped to elements that contain the `nillable="true"` attribute.
- The complex type used to define the type of the input message GED shall contain all and only those elements that correspond to method parameters that have their `In` qualifier effectively defined as `true`. The sequence of these elements in the complex type shall correspond to the sequence of the input parameters in the CIM definition of the method.

  If the method has no input parameters, the complex type used in the GED shall be empty (that is, `<xs:complexType/>`).
- The complex type used to define the type of the output message GED shall contain all elements that correspond to method parameters that have their `Out` qualifier effectively defined as `true`. The sequence of these elements in the complex type shall correspond to the sequence of the output parameters in the CIM definition of the method.
- The complex type used to define the output message GED shall contain an element of `name="ReturnValue"` as the final element in its sequence. The following rules govern the structure of this element:
  - The XSD type of this element shall be mapped from the CIM method type in compliance with the datatype conversion defined in clause 8.
  - The element shall include the attribute `nillable="true"`. (See the following note.)
- Parameters and return values may be defined in CIM with `ValueMap` qualifiers. Mapping these `ValueMaps` to metadata fragments is required (see 11.1). In addition, a `ValueMap` shall be mapped to an enumeration/union associated with the mapped parameter or return value in the XSD (see 11.2). The mapping shall conform to the rules for mapping `ValueMaps` described in 9.2.3.

Notes on the preceding rules:

- A parameter shall occur in both the complex types used to define the input and output message GEDs if it has both the `In` and `Out` qualifiers effectively defined as `true`.

DSP0004 allows NULL as the default return value for all methods. Thus, this specification must allow for the possibility that the return value of any method may be NULL.

9.5.2 Inheritance of Methods

Classes may inherit some of the methods that they own. In general, classes inherit all methods specified in or inherited by their superclasses, along with all qualifiers that are specified as `ToSubClass`. The `Override` qualifier, however, determines special behavioral considerations on the part of conversion algorithms. The following rules govern the inheritance of methods under conditions of override:

- When multiple methods in the inheritance chain that have the same name are not overridden, the method in the most derived class shall be mapped. Any other duplicate, but not overridden, methods shall not be mapped.

- When a method in a derived class overrides another method (of the same name and signature) in a superclass, the method in the most derived class shall be mapped, including all qualifiers inherited from the overridden methods. The overridden method shall not be mapped.
- The inheritance of qualifiers pertaining to methods shall comply with the inheritance rules regarding qualifiers in C.3.

EXAMPLE: As an example of the preceding rules, consider the following MOF method definition extracted from the example in C.1.2. (See C.2.2 for the complete example.)

```
class EX_DerivedComponent
{
...
uint32 RequestStateChange([IN] uint16 RequestedState, [OUT] [IN(False)] CIM_SomeClass REF
ResultClass, [IN] datetime TimeoutPeriod);
};
```

The input parameters, designated in the MOF by the `In` qualifier, would be mapped as follows:

```
<xs:element name="RequestStateChange_INPUT">
<xs:complexType>
      <xs:sequence>
        <xs:element name="RequestedState" type="cim:cimUnsignedShort"
                 nillable="true"/>
        <xs:element name="TimeoutPeriod" type="cim:cimDateTime"
                 nillable="true"/>
       </xs:sequence>
  </xs:complexType>
</xs:element>
```

The output parameters, designated in the MOF by the `Out` qualifier, would be mapped in the following way. Note that the complex type includes an element that represents the return value of the CIM method.

```
<xs:element name="RequestStateChange_OUTPUT">
<xs:complexType>
   <xs:sequence>
          <xs:element name="ResultClass" type="cim:cimReference"
                 nillable="true"/>
       <xs:element name="ReturnValue" type="cim:cimUnsignedInt"
                 nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

9.5.3 Parameters and Return Values with ValueMaps

Input and output parameters and return values of the method may be specified with a `ValueMap` qualifier. The `ValueMap` shall be mapped to the XSD element that represents the parameter or return value.

The `RequestedState` input parameter must be defined as an enumeration in accordance with its `ValueMap` (see C.1.2 for this `ValueMap`):

```
<xs:element name="RequestedState" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="cim:cimUnsignedShort">
        <xs:enumeration value="2"/>
        <xs:enumeration value="3"/>
        <xs:enumeration value="4"/>
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

9.6 CIM Instances

This clause describes the representation of CIM instances. The intended scope is all representations of CIM instances used in any protocols.

CIM instances are represented according to the following rules:

- Representations of CIM instances shall be XML instance documents that conform to the XSD schema for their CIM creation class, as defined in clause 9.
- The class namespace used within an instance document shall be a namespace URI and it shall be defined as follows:

  'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-major-version '/' cim-class-schema '_' cim-class-name

  Where:

  – wscim-major-version is the major version number of this specification. Note that this version number changes only if there are incompatible changes in the specification.

  – cim-schema-major-version is the major version number of the CIM schema version to which the class being converted belongs. Note that this version number changes only if there are incompatible changes in the CIM schema.
- The location of the specific schema used to construct the instance should be declared by use of the xsi:schemaLocation attribute where the namespace URI and the specific class schema document URI location are combined as the value of the attribute, separated by whitespace. This provides a means for a recipient of the instance to determine which version of CIM defines the structure of this instance.

  For example:

  If the instance document is constructed under CIM schema version 2.11.0 and WS-CIM mapping specification 1.3.0, the following xsi:schemaLocation attribute should be specified in the instance document (where *ClassName* is the name of the CIM class of the instance):

  ```
  xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/ClassName 
  http://schemas.dmtf.org/wbem/wscim/1.3.0/cim-schema/2.11.0/ClassName.xsd"
  ```

If the URI in the attribute value can be de-referenced, then strict XML schema validation can be achieved.

9.7 Superclass Class Representation

The CIM Schema defines CIM classes in subclass / superclass relationships (hierarchies). For example, MOF reflects this structure in the definition of the class. A MOF statement of the following form defines CIM_ClassA as a subclass of the superclass CIM_ClassB:

```
class CIM_ClassA : CIM_ClassB
```

A MOF statement of the following form defines CIM_ClassC as a top-level class with no superclass:

```
class CIM_ClassC
```

Some management protocols may require a representation of the subclass/superclass hierarchy of a class as a value of an XML element in instance documents. The XSD mechanism by which to support representing this structure as a value of an XML element is provided in a separate schema that may be imported by protocols that require this capability for their instance documents.

The immediate subclass/superclass relationship of a class shall be mapped to a single GED in the classhierarchy namespace according to the following rules:

- Elements that describe the subclass / superclass relationships shall be placed in a separate schema from the WS-CIM class schema. For subclass / superclass relationships defined in the CIM Schema, the schema shall be named as follows:

  'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-major-version '/classhierarchy'

  Where:

  – wscim-major-version is the major version number of this specification. Note that this version number changes only if there are incompatible changes in the specification.

  – cim-schema-major-version is the major version number of the CIM schema version to which the class being converted belongs. Note that this version number changes only if there are incompatible changes in the CIM schema.

This schema shall import the http://schemas.dmtf.org/wbem/wscim/*n*/classhiertype namespace, where '*n*' represents the version number of this namespace.

The class being mapped shall be represented by a GED whose name is derived from the name of the CIM class, appended with "`_Class`". This element shall be defined by an anonymous complex type that follows one of the following patterns:

- The WS-CIM schema of a top-level class, XXX_ClassC, shall define the `_Class` element as a restriction of the `ctype:ClassHierarchyType`. The following pattern illustrates this rule:

```
<xs:element name="XXX_ClassC_Class">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="ctype:ClassHierarchyType" />
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

- The WS-CIM schema of a subclass, XXX_ClassA, that is derived by a superclass, XXX_ClassB, shall restrict the `ctype:classHierarchyType` to contain a single element that references the corresponding `_Class` GED of the superclass. The following pattern illustrates this rule:

<xs:element name="XXX_ClassA_Class">

```
    <xs:complexType>
        <xs:complexContent>
      <xs:restriction base="ctype:ClassHierarchyType">
         <xs:sequence>
           <xs:element ref="chier:XXX_ClassB_Class"/>
         </xs:sequence>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

See C.2.4 for an example classhierarchy schema.

10 CIM Methods to WSDL Mappings

This clause defines the structures that are necessary to define the messages and operation structures required for mapping a CIM method to WSDL.

10.1 Defining WSDL Message Structures

This clause provides the rules for creating WSDL message structures.

The rules that govern the creation of WSDL message structures for a method are as follows:

- The `name` of the WSDL input message should be the name of the CIM method plus `_InputMessage`. The following rules specify the structure of this element:
  - The `name` of the `wsdl:part` element should be "`body`".
  - The `element` attribute of the `wsdl:part` element shall specify the QName of the input message GED for the CIM method (see 9.5).
- The `name` of the WSDL output message should be the name of the CIM method plus `_OutputMessage`. The following rules specify the structure of this element:
  - The `name` of the `wsdl:part` element should be "`body`".
  - The `element` attribute of the `wsdl:part` element shall specify the QName of the output message GED defined for the CIM method (see 9.5).

EXAMPLE: The `wsdl:message` elements for the `RequestStateChange` CIM method (see 9.5) would be specified in the WSDL document as follows. The `wsdl:message` intended for input to the WSDL operation would be as follows (where the "`class:`" namespace prefix represents the namespace of the class whose interface is being exposed through this WSDL):

```
<wsdl:message name="RequestStateChange_InputMessage">
  <wsdl:part name="body"
    element="class:RequestStateChange_INPUT"/>
</wsdl:message>
```

See C.3 for an example that shows the complete specification of the WSDL messages for this operation.

10.2 Defining WSDL Operation Structures

This specification defines *only* the structure of WSDL `portType` operations.

The rules governing the structure of the WSDL operations used to invoke CIM methods are as follows:

- The `name` attribute of the `wsdl:operation` element shall be the name of the CIM method.
- The `name` attributes of the `wsdl:input` and `wsdl:output` child elements should be the `name` of the `wsdl:messages` that are referenced by these elements.
- The `message` attributes of the `wsdl:input` and `wsdl:output` elements shall specify the QName of input and output message elements defined in 10.1.

EXAMPLE: The `RequestStateChange` CIM method (see 9.5) is defined as follows:

```
<wsdl:definitions
  ...
  xmlns:thisWSDL="http://. . ..wsdl"
  ...>
<wsdl:operation name="RequestStateChange">
    <wsdl:input name="RequestStateChange_InputMessage"
      message="thisWSDL:RequestStateChange_InputMessage"/>
    <wsdl:output name="RequestStateChange_OutputMessage"
      message="thisWSDL:RequestStateChange_OutputMessage"/>
</wsdl:operation>
</wsdl:definitions>
```

This definition should be included in the `wsdl:portType` section of a WSDL document of a service that makes the CIM `RequestStateChange` method available to clients.

10.3 Defining wsa:Actions

The Addressing specifications (*Web Services Addressing (WS-Addressing) 1.0 – Core* and DSP0226, "Management Addressing" clause) define the information model and syntax for the abstract messaging property Action. This property is defined as an IRI (RFC 3987), which identifies the semantics implied by a message (input, output, or fault). For the purposes of this specification, the Action property is restricted to a URI (RFC 3986) (a sequence of characters from a limited subset of the repertoire of US-ASCII characters).

The details of how the action URI is specified on description artifacts are left to the specific protocol-binding specifications. Action URIs for faults are always left to the protocol-binding specifications.

The rules for constructing WSA action URIs for input and output operation elements are as follows:

- The action URI for an input message shall have the following form:

  class-namespace-name '/' input-name

  Where:

  – class-namespace-name is the full namespace name of the class being mapped as defined in 9.1.

  – input-name is the name of the CIM method.

- The action URI for an output message shall have the following form:

  class-namespace-name '/' output-name ‘Response’

  Where:

  – class-namespace-name is the full namespace name of the class being mapped as defined in 9.1.

  – output-name is the name of the output CIM method.

EXAMPLE: The action URI for the input message of the `RequestStateChange` method is as follows:

```
wsa:Action="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/EX_DerivedComponent/
RequestStateChange"
```

The action URI for the output message of the `RequestStateChange` method is as follows:

wsa:Action="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent/
RequestStateChangeResponse"

See C.3 for an example that shows a complete mapping of the `RequestStateChange` method.

11 Qualifier Mappings

This clause defines the mapping of qualifiers to metadata fragments. The definition of the container for metadata fragments is left to the specific protocols to specify.

11.1 General Format

Rules for mapping qualifiers fall into two categories:

- rules that describe the type definitions of qualifiers
- rules that describe the XSD elements to which qualifiers are mapped

This specification provides type definitions for the qualifier types used in CIM in common.xsd and mappings for all CIM qualifiers in the qualifiers.xsd (Annex A.2). It is expected that user-defined qualifiers follow the mapping rules for mapping qualifiers described in the following clauses.

In addition to the mapping rules, user-defined qualifier mappings are governed by the following rules:

- User-defined qualifiers shall not use any qualifier namespace defined in this specification.

The qualifier types (types with names of the qualifier*Type*) defined in common.xsd should be used to define user-defined qualifiers. In the majority of cases, it is not necessary to create a new type definition to define a qualifier.

11.1.1 Single-Valued Qualifiers

This clause describes the rules for mapping single-valued qualifiers.

Single-valued qualifiers that have an effective value that matches their default value shall not be mapped to a metadata fragment.

The rules for mapping single-valued qualifiers that have an effective value that is not their global default value are as follows:

- The rules for defining the type of a single-valued qualifier are as follows:
  - Single-valued qualifiers shall be mapped to a complex type that is an extension of a simple content.
  - The base type of this complex type shall correspond to the type of qualifier according to the datatype conversion rules defined in clause 8.
  - The complex type shall extend the base type with a Boolean attribute of the name `qualifier`. This attribute is defined in the cim namespace. This attribute shall be specified with the XML Schema attribute `use="required"`.
- The rules for mapping a single-valued qualifier are as follows:
  - The qualifier shall be mapped to a GED whose type corresponds to the datatype type of the qualifier and which has been defined by the preceding rule. The name of the element shall be the name of the qualifier.
  - The value of the cim:`qualifier` attribute in the mapped metadata fragment shall be `true` in all mappings.
  - Single-valued qualifiers implicitly have the multiplicity `minOccurs="0" maxOccurs="1"`. Protocols should provide a mechanism by which to express this multiplicity in the schema of the document that contains generated metadata fragments.

- The format of a schema element for defining a single-valued qualifier is as follows:

  ```
  '<xs:element name="' cim:qualifier-name '" type="' qualifier-type '"/>''
  ```

  Where:

  – qualifier-type is typically one of the qualifier*Type* types defined in common.xsd (but may be a user-defined type that complies with the mapping rules).

cim:qualifier-name is the name of the qualifier, qualified by its namespace prefix.

EXAMPLE: A generalized example of the mapping of a single-valued qualifier is as follows:

```
<ns:QualifierName cim:qualifier="true">
    value
</ns:QualifierName>
```

Where:

– QualifierName is the name of the qualifier.

– ns is the namespace prefix referencing the namespace in which this qualifier is defined.

– value is the string representation of the qualifier value.

For example, the mapping of the CIM qualifier `Abstract` is as follows:

```
<cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
```

11.1.2 Multi-Valued Qualifiers

This clause describes the rules for mapping multi-valued qualifiers.

Multi-valued qualifiers are qualifiers that are arrays. Multi-valued qualifiers that have an effective value that matches their default value shall not be mapped to a metadata fragment.

The rules for mapping multi-valued qualifiers that have an effective value that is not their global default value are as follows:

- The rules for defining the type of a multi-valued qualifier are as follows:
  – Multi-valued qualifiers shall be mapped to a complex type that is an extension of a complex content.
  – The base type of this complex type shall correspond to the single-valued qualifier*Type* of the member elements from which the qualifier array is constructed.
- The rules for mapping a multi-valued qualifier are as follows:
  – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of the qualifier and which has been defined by the preceding rule. The `name` of the element shall be the same as that of the qualifier itself.
  – The value of the cim:`qualifier` attribute in the metadata fragments shall be `true` in all mappings.
  – Multi-valued qualifiers implicitly have the default multiplicity `minOccurs="0" maxOccurs="unbounded"`. Qualifier declarations may explicitly set different bounds on an array qualifier. Protocols should provide a mechanism by which to express the size of an array qualifier in the schema of the document that contains generated metadata fragments. The `minOccurs` and `maxOccurs` values shall correspond either to the default values if no qualifier array size is declared or to the declared qualifier array size.

NOTE: The common.xsd file defines a string array of qualifier values, `qualifierSArray`. This definition complies with the first mapping rule in this clause. Therefore, in the majority of cases, it is not necessary to create a new array complex type definition to define a multi-valued qualifier. Rather, it is sufficient to use the `qualifierSArray` type defined in common.xsd.

The format for a schema for defining a multi-valued qualifier is as follows:

```
'<xs:element name="' cim:qualifier-name '" type="' qualifier-array-type ‘"/>’'
```

Where:

- cim:qualifier-name is the name of the qualifier, qualified by its namespace prefix.
- qualifier-array-type is typically the `qualifierSArray` type defined in the common.xsd file (but may be a user-defined type that complies with the mapping rules).

EXAMPLE 1: A generalized example of the mapping of a multi-valued qualifier is as follows:

```
<ns:QualifierName cim:qualifier="true">
    value
</ns:QualifierName>
...    // repeat QualifierName elements for each member of the array
```

Where:

- QualifierName is the name of the qualifier.
- ns is the namespace prefix referencing the namespace in which this qualifier is defined.
- n is a sequential integer that represents the position of the entry in the array.
- value is the string representation of the qualifier value.

EXAMPLE 2: For example, the mapping of a `ValueMap` qualifier containing three entries `( "OK", "Error", "Unknown" )` is as follows:

```
<cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
```

A complete mapping of all qualifiers is provided in qualifiers.xsd (Annex A.2).

11.2 Mapping CIM Qualifiers to XSD Elements

All qualifiers are mapped using the normative rules in 11.1. The qualifiers listed in Table 8 are also mapped directly into XSD features.

Table 8 – CIM Qualifiers Mapped to XSD Elements

CIM Qualifier	MOF Example	Mapped to XSD Structure
`Embedded Instance`	`[EmbeddedInstance ("Class" )]`	`xs:anyType` (normatively defined in 9.2.5.1)
`Embedded Object`	`[EmbeddedObject]`	`xs:anyType` (normatively defined in 9.2.5.1)
`Key`	`[Key]`	`nillable="false"` (normatively defined in 9.2.1.1) NOTE: `False` is the default value of the `nillable` attribute and therefore may not be explicitly expressed in the schema.
`IN`	`[IN] / [IN ( true )]`	The CIM input parameter is mapped to an element in the complex type for the input message GED (normatively defined in 9.5.1).

CIM Qualifier	MOF Example	Mapped to XSD Structure

`MaxLen`	`[MaxLen ( 1024 )]`	Mapped to a restriction using `xs:maxLength` on a string datatype. Required, with the following exception:
		A qualifier value of `NULL` shall not be mapped.
		For example:
		`<xs:element name="`*`PropName`*`">`
		  `<xs:complexType>`
		    `<xs:simpleContent>`
		     `<xs:restriction base="cim:cimString">`
		       `<xs:maxLength value="1024"/>`
		       `<xs:anyAttribute .../>`
		     `</xs:restriction>`
		     `</xs:simpleContent>`
		  `</xs:complexType>`
		`</xs:element>`

`MaxValue`	`[MaxValue ( 100 )]`	Mapped to a restriction using `xs:maxInclusive` on an integer datatype. Required, with the following exception:
		A qualifier value of `NULL`, which indicates the largest value allowed by the type, should not be mapped.
		For example:
		`<xs:element name="`*`PropName`*`">`
		  `<xs:complexType>`
		    `<xs:simpleContent>`
		     `<xs:restriction base="cim:cimUnsignedShort">`
		       `<xs:maxInclusive value="100"/>`
		       `<xs:anyAttribute ... />`
		     `</xs:restriction>`
		    `</xs:simpleContent>`
		  `</xs:complexType>`
		`</xs:element>`

`MinLen`	`[MinLen ( 10 )]`	Mapped to a restriction using `xs:minLength` on a string datatype. Required, with the following exception:
		A qualifier value of `0` should not be mapped.
		For example:
		`<xs:element name="`*`PropName`*`">`
		  `<xs:complexType>`
		    `<xs:simpleContent>`
		      `<xs:restriction base="cim:cimString">`
		        `<xs:minLength value="10"/>`
		        `<xs:anyAttribute ... />`
		      `</xs:restriction>`
		     `</xs:simpleContent>`
		  `</xs:complexType>`
		`</xs:element>`

CIM Qualifier	MOF Example	Mapped to XSD Structure

`MinValue`	`[MinValue ( 10 )]`	Mapped to a restriction using `xs:minInclusive` on an integer datatype. Required, with the following exception:
		A qualifier value of `NULL`, which indicates the smallest value allowed by the type, should not be mapped.
		For example:
		`<xs:element name="PropName">`
		`  <xs:complexType>`
		`    <xs:simpleContent>`
		`      <xs:restriction base="cim:cimUnsignedShort">`
		`        **<xs:minInclusive value="10"/>**`
		`        <xs:anyAttribute . . ./>`
		`      </xs:restriction>`
		`    </xs:simpleContent>`
		`  </xs:complexType>`
		`</xs:element>`

`OctetString`
`uint8[]`
`string[]`
	`[OctetString]`	Normatively defined in 9.2.4.1
		`cim:cimBase64Binary`
		`cim:cimHexBinary array`

`OUT`	`[OUT]`	The CIM output parameter is mapped to an element in the complex type for the output message GED (normatively defined in 9.5.1).
`Override`	`[Override ( "PropName" )]`	Determines the behavior of the mapping algorithm: a mapping shall select the most derived property, reference, or method for inclusion in a derived class (normatively defined in 9.4 and 9.5.2). The following rules govern specific behavior regarding the inheritance of qualifiers (normatively defined in 11.3): • For non-overridden properties, `Override(NULL)`, only the qualifiers in the most derived class shall be mapped. • For overridden properties, the effective values of inherited qualifiers shall be considered in the mapping.
`Required`	`[Required]`	`nillable="false"` (normatively defined in 9.2.1.1) NOTE: If the effective value of the `Required` qualifier is false, then `nillable="true"` (required).
`ValueMap`	`[ValueMap (...)]`	ValueMaps may be mapped to XSD as an enumeration (see 9.2.3).

11.3 Inheritance of Qualifiers

In addition to inheritance of properties, references, and methods through class inheritance, qualifier values on any CIM elements are inherited. However, qualifiers are subject to special rules of inheritance. Qualifier inheritance behavior is defined by the Flavors associated with a particular qualifier declaration.

The rules covering qualifier inheritance are summarized in the third column of Table 9. In Table 9, the term "overriding CIM elements in any subclasses" refers to CIM properties, references, and methods that override other occurrences of the properties, references, or methods in their superclasses and therefore form an inheritance chain. Note that duplicate property, reference, or method names that are *not* overridden interrupt the inheritance chain for these CIM elements to their superclasses.

1436 **Table 9 – Rules of Qualifier Inheritance**

FLAVOR	Qualifier Inheritance Behavior (Informative)	Metadata Fragment Mapping Behavior
`Restricted`	The qualifier value pertains only to the CIM element on which it is defined. It is not inherited by any subclasses or overriding CIM elements in these subclasses. EXAMPLE: `Abstract`	The metadata fragment mapping for the qualifier applies only to the XSD element mapped from the CIM element that has the qualifier value defined. The metadata fragment shall not be carried to corresponding CIM elements in any subclasses.
`ToSubclass: EnableOverride`	The qualifier value is inherited by any subclasses or overriding CIM elements in these subclasses. The value of the qualifier may be changed in a subclass. EXAMPLE: `MaxLen`	The metadata fragment mapping for the qualifier applies to the XSD element mapped from the CIM element that has the qualifier value defined. In addition, the metadata fragment shall be carried to any subclasses or overriding CIM elements in these subclasses. In addition, the metadata fragment shall reflect the qualifier value provided on the corresponding CIM element. Unless overridden on the corresponding CIM element, the metadata fragment shall have the same value as the defined value in the superclass.
`ToSubclass: DisableOverride`	The qualifier value is inherited by any subclasses or overriding CIM elements in these subclasses. The value of the qualifier must not be changed in a subclass. EXAMPLE: `Key`	The metadata fragment mapping for the qualifier applies to the XSD element mapped from the CIM element that has the qualifier value defined. In addition, the metadata fragment shall be carried to any subclasses or overriding CIM elements in these subclasses.

ANNEX A (Informative)

Schemas

This annex provides examples of the WS-CIM Schema (DSP8004), the Qualifiers Schema (DSP8005), and the Class Hierarchy Type Schema (DSP8006).

A.1 Common WS-CIM Schema: DSP8004

This schema contains common definitions.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

<!--  The following are runtime attribute definitions -->
  <xs:attribute name="Key" type="xs:boolean"/>

  <xs:attribute name="Version" type="xs:string"/>

<!--  The following section defines the extended WS-CIM datatypes  -->

  <xs:complexType name="cimDateTime">
    <xs:choice>
      <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
      <xs:element name="Interval" type="xs:duration"/>
      <xs:element name="Date" type="xs:date" />
      <xs:element name="Time" type="xs:time" />
      <xs:element name="Datetime" type="xs:dateTime"/>
    </xs:choice>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="cimUnsignedByte">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedByte">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
     </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimByte">
    <xs:simpleContent>
      <xs:extension base="xs:byte">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedShort">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedShort">
```

```
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimShort">
    <xs:simpleContent>
      <xs:extension base="xs:short">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedInt">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedInt">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimInt">
    <xs:simpleContent>
      <xs:extension base="xs:int">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimUnsignedLong">
    <xs:simpleContent>
      <xs:extension base="xs:unsignedLong">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimLong">
    <xs:simpleContent>
      <xs:extension base="xs:long">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimString">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimBoolean">
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimFloat">
    <xs:simpleContent>
      <xs:extension base="xs:float">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimDouble">
    <xs:simpleContent>
      <xs:extension base="xs:double">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
  </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimChar16">
    <xs:simpleContent>
      <xs:restriction base="cim:cimString">
         <xs:maxLength value="1"/>
         <xs:anyAttribute namespace="##any" processContents="lax"/>
       </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimBase64Binary">
    <xs:simpleContent>
      <xs:extension base="xs:base64Binary">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimHexBinary">
    <xs:simpleContent>
      <xs:extension base="xs:hexBinary">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="cimAnySimpleType">
    <xs:simpleContent>
      <xs:extension base="xs:anySimpleType">
         <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

<xs:complexType name="cimReference">
  <xs:sequence>
    <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
```

```

<!--  The following datatypes are used exclusively to define metadata fragments  -->
  <xs:attribute name="qualifier" type="xs:boolean"/>

  <xs:complexType name="qualifierString">
    <xs:simpleContent>
      <xs:extension base="cim:cimString">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierBoolean">
    <xs:simpleContent>
      <xs:extension base="cim:cimBoolean">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierUInt32">
    <xs:simpleContent>
      <xs:extension base="cim:cimUnsignedInt">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierSInt64">
    <xs:simpleContent>
      <xs:extension base="cim:cimLong">
        <xs:attribute ref="cim:qualifier" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="qualifierSArray">
    <xs:complexContent>
      <xs:extension base="cim:qualifierString"/>
    </xs:complexContent>
  </xs:complexType>

<!--  The following element is to be used only for defining metadata -->
  <xs:element name="DefaultValue" type="xs:anySimpleType" />

</xs:schema>
```

A.2 Qualifiers Schema: DSP8005

The following schema is an example of the qualifiers schema that is based on CIM Schema 2.13.1. Future versions of CIM Schema may add or delete qualifiers, which would be reflected in the corresponding qualifiers.xsd file.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
targetNamespace="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
    xmlns:cimQ="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
    xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

  <xs:import
     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>

  <xs:element name="Abstract" type="cim:qualifierBoolean"/>
  <xs:element name="Aggregate" type="cim:qualifierBoolean"/>
  <xs:element name="Aggregation" type="cim:qualifierBoolean"/>
  <xs:element name="ArrayType" type="cim:qualifierString"/>
  <xs:element name="Association" type="cim:qualifierBoolean"/>
  <xs:element name="BitMap" type="cim:qualifierSArray"/>
  <xs:element name="BitValues" type="cim:qualifierSArray"/>
  <xs:element name="ClassConstraint" type="cim:qualifierSArray"/>
  <xs:element name="Counter" type="cim:qualifierBoolean"/>
  <xs:element name="Composition" type="cim:qualifierBoolean"/>
  <xs:element name="Deprecated" type="cim:qualifierSArray"/>
  <xs:element name="Description" type="cim:qualifierString"/>
  <xs:element name="DisplayName" type="cim:qualifierString"/>
  <xs:element name="DN" type="cim:qualifierBoolean"/>
  <xs:element name="EmbeddedInstance" type="cim:qualifierBoolean"/>
  <xs:element name="EmbeddedObject" type="cim:qualifierBoolean"/>
  <xs:element name="Exception" type="cim:qualifierBoolean"/>
  <xs:element name="Experimental" type="cim:qualifierBoolean"/>
  <xs:element name="Gauge" type="cim:qualifierBoolean"/>
  <xs:element name="In" type="cim:qualifierBoolean"/>
  <xs:element name="Indication" type="cim:qualifierBoolean"/>
  <xs:element name="Key" type="cim:qualifierBoolean"/>
  <xs:element name="MappingStrings" type="cim:qualifierSArray"/>
  <xs:element name="Max" type="cim:qualifierUInt32"/>
  <xs:element name="MethodConstraint" type="cim:qualifierSArray"/>
  <xs:element name="Min" type="cim:qualifierUInt32"/>
  <xs:element name="MaxLen" type="cim:qualifierUInt32"/>
  <xs:element name="MaxValue" type="cim:qualifierSInt64"/>
  <xs:element name="MinLen" type="cim:qualifierUInt32"/>
  <xs:element name="MinValue" type="cim:qualifierSInt64"/>
  <xs:element name="Revision" type="cim:qualifierString"/>          <!--  Is Deprecated  -->
  <xs:element name="ModelCorrespondence" type="cim:qualifierSArray"/>
  <xs:element name="NullValue" type="cim:qualifierString"/>
  <xs:element name="OctetString" type="cim:qualifierBoolean"/>
  <xs:element name="Out" type="cim:qualifierBoolean"/>
  <xs:element name="Override" type="cim:qualifierString"/>
  <xs:element name="Propagated" type="cim:qualifierString"/>
  <xs:element name="PropertyConstraint" type="cim:qualifierSArray"/>
  <xs:element name="Read" type="cim:qualifierBoolean"/>
  <xs:element name="Required" type="cim:qualifierBoolean"/>
  <xs:element name="Schema" type="cim:qualifierString"/>
  <xs:element name="Static" type="cim:qualifierBoolean"/>
  <xs:element name="Terminal" type="cim:qualifierBoolean"/>
  <xs:element name="Units" type="cim:qualifierString"/>
  <xs:element name="UMLPackagePath" type="cim:qualifierString"/>
  <xs:element name="ValueMap" type="cim:qualifierSArray"/>
  <xs:element name="Values" type="cim:qualifierSArray"/>
  <xs:element name="Version" type="cim:qualifierString"/>
  <xs:element name="Weak" type="cim:qualifierBoolean"/>
  <xs:element name="Write" type="cim:qualifierBoolean"/>

<!--  Qualifier defined by DMTF for a future release of CIM Schema             -->
<!--  Included in this version at the request of the WSDM-CIM mapping team     -->
  <xs:element name="Correlatable" type="cim:qualifierSArray"/>

<!--  Following qualifiers are considered to be "Optional Qualifiers" in CIM.  -->
  <xs:element name="Alias" type="cim:qualifierString"/>
  <xs:element name="Delete" type="cim:qualifierBoolean"/>
  <xs:element name="Expensive" type="cim:qualifierBoolean"/>
  <xs:element name="IfDeleted" type="cim:qualifierBoolean"/>
  <xs:element name="Invisible" type="cim:qualifierBoolean"/>
  <xs:element name="Large" type="cim:qualifierBoolean"/>
  <xs:element name="Provider" type="cim:qualifierString"/>
  <xs:element name="PropertyUsage" type="cim:qualifierString"/>
  <xs:element name="Syntax" type="cim:qualifierString"/>
  <xs:element name="SyntaxType" type="cim:qualifierString"/>
  <xs:element name="TriggerType" type="cim:qualifierString"/>
  <xs:element name="UnknownValues" type="cim:qualifierSArray"/>
  <xs:element name="UnsupportedValues" type="cim:qualifierSArray"/>

</xs:schema>
```

A.3 Class Hierarchy Type Schema: DSP8006

The complex type definition in the following schema provides the type of GEDs that describe the CIM Schema subclass / superclass hierarchy. The element `ClassHierarchy` may be used by protocols as an element in XML instance documents of a CIM instance that contains a value representing the subclass / superclass hierarchy of a class. Its presence as an element in an instance document would be covered by the `xs:any` in the WS-CIM schema of the instance's class.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
    xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="ClassHierarchyType">
    <xs:sequence>
      <xs:any minOccurs="0" namespace="##any" processContents="lax" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="ClassHierarchy" type="ctype:ClassHierarchyType"/>

</xs:schema>
```

ANNEX B
(Informative)

Conventions

In XML and MOF examples, an ellipsis (`"..."`) indicates omitted or optional entries that would typically occupy the position of the ellipsis.

The following conventions are followed for defining formats of entries such as URIs:

- Literal characters within a format definition are surrounded by single quotes.
- Names of variables within a format are in standard text and are explicitly defined by means of a "Where: variable-name is …" section that follows the format definition.
- A specific value of a variable within a generalized example of a formatted entry is displayed in *italics*.
- Definitions of formats are case sensitive.
- Whitespace, if any, in formats is explicitly indicated.

The following typographical conventions are used:

- `Monospace font`: CIM datatypes and element names as well as XML and WSDL element and attribute names.
- `Courier new 8, gray background`: Code examples

ANNEX C
(Informative)

Examples

This annex contains examples of converting MOF definitions of several classes into XML Schema, WSDL fragments, and metadata fragments. Although the classes are fictional creations used to illustrate different features of the conversion, the classes are based on actual CIM classes.

C.1 MOF Definitions

This clause contains the MOF definitions that are converted in these examples.

C.1.1 EX_BaseComponent

```
   [Abstract, Version ( "2.x" ), Description (
       "EX_BaseComponent serves as an example base CIM class.")]
class EX_BaseComponent {
      [Description (
          "A datetime value indicating when the object was installed.")]
   datetime InstallDate;
      [Description (
          "The Name property defines the label by which the object is "
          "known."),
       MaxLen ( 1024 ), Required]
   string Name;
      [Description (
          "A set of descriptive statements that can be used to describe the "
          "state of a Component."),
       ArrayType ( "Indexed" ) ]
   string StatusDescriptions[];
      [Description (
          "A descriptive code representing operational health of a Component."),
       ValueMap { "OK", "Error", "Unknown" }, MaxLen ( 10 )]
   string HealthStatus;
};
```

C.1.2 EX_DerivedComponent

```
   [Version ( "2.x" ), Description (
       "This class extends EX_BaseComponent.")]
class EX_DerivedComponent : EX_BaseComponent {
      [Description (
          "EnabledState is an integer enumeration that indicates the "
          "enabled and disabled states of a derived Component."),
       ValueMap { "0", "1", "2", "3" },
       Values { "Unknown", "Other", "Enabled", "Disabled" } ]
   uint16 EnabledState=3;
      [Description (
          "Boolean flag indicating availability of a Component.") ]
   boolean AvailableFlag;
      [Description (
          "Requests that the state of the element be changed to the "
          "value specified in the RequestedState parameter."),
       ValueMap { "0", "1", "2", "3..32767", "32768..65535" },
       Values { "Completed with No Error", "Not Supported",
                "Failed", "DMTF Reserved", "Vendor Specific" } ]
   uint32 RequestStateChange(
         [IN, Description (
             "The state requested for the Component."),
          ValueMap { "2", "3", "4" },
          Values { "Enabled", "Disabled" "Shutdown" } ]
      uint16 RequestedState,
         [IN ( false ), OUT, Description (
             "Reference to an instance of some class (undefined in this "
             "example) that is returned upon completion of the operation.")]
      CIM_SomeClass REF ResultClass,
         [IN, Description (
             "A timeout period that specifies the maximum amount of "
             "time that the client expects the transition to the new "
             "state to take. ")]
      datetime TimeoutPeriod);
};
```

C.1.3 EX_AssociationComponent

```
   [Association, Version ( "2.x" ), Description (
       "Indicates that two entites are associated.")]
class EX_Association {
      [Key, Description (
          "AssociatingComponent represents one Component is "
          "associated with the Component referenced as AssociatedComponent.")]
   EX_BaseComponent REF AssociatingComponent;
      [Key, Description (
          "AssociatedComponent represents another Component (up to 4) that "
          "is associated with the Component referenced as "
          "AssociatingComponent."),
          Max ( 4 )]
   EX_BaseComponent REF AssociatedComponent;
      [Description (
          "The point in time that the Components were associated.")]
   datetime WhenAssociated;
      [Description (
          "Boolean indicating whether the association is maintained.")]
   boolean AssocMaintained;
};
```

C.2 XSD

This clause shows the XML Schema files that would result from the application of this specification to the preceding example CIM classes.

C.2.1 EX_BaseComponent

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
    xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
    xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    …>
  <xs:import
     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
   <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
```

```
    <xs:element name="Name">
      <xs:complexType>
        <xs:simpleContent>
           <xs:restriction base="cim:cimString">
              <xs:maxLength value="1024"/>
              <xs:anyAttribute namespace="##any" processContents="lax"/>
           </xs:restriction>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="StatusDescriptions" type="cim:cimString"/>
    <xs:element name="HealthStatus" nillable="true">
      <xs:complexType>
        <xs:simpleContent>
          <xs:restriction base="cim:cimString">.
             <xs:enumeration value="OK"/>
             <xs:enumeration value="Error"/>
             <xs:enumeration value="Unknown"/>
             <xs:maxLength value="10"/>
             <xs:anyAttribute namespace="##any" processContents="lax"/>
          </xs:restriction>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
   <xs:complexType name="EX_BaseComponent_Type">
     <xs:sequence>
       <xs:element ref="class:HealthStatus" minOccurs="0"/>
       <xs:element ref="class:InstallDate" minOccurs="0"/>
       <xs:element ref="class:Name"/>
       <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:anyAttribute namespace="##any" processContent="lax"/>
   </xs:complexType>
   <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type"/>
</xs:schema>
```

C.2.2 EX_DerivedComponent

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
    xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
    xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    ...>
  <xs:import
     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
   <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
   <xs:element name="Name">
     <xs:complexType>
       <xs:simpleContent>
          <xs:restriction base="cim:cimString">
             <xs:maxLength value="1024"/>
             <xs:anyAttribute namespace="##any" processContents="lax"/>
          </xs:restriction>
       </xs:simpleContent>
     <xs:complexType>
   </xs:element>
  <xs:element name="StatusDescriptions" type="cim:cimString"/>
   <xs:element name="HealthStatus" nillable="true">
     <xs:complexType>
       <xs:simpleContent>
         <xs:restriction base="cim:cimString">
            <xs:enumeration value="OK"/>
            <xs:enumeration value="Error"/>
            <xs:enumeration value="Unknown"/>
            <xs:maxLength value="10"/>
           <xs:anyAttribute namespace="##any" processContents="lax"/>
         </xs:restriction>
       </xs:simpleContent>
     </xs:complexType>
   </xs:element>
   <xs:element name="EnabledState" nillable="true">
     <xs:complexType>
       <xs:simpleContent>
         <xs:restriction base="cim:cimUnsignedShort">
           <xs:enumeration value="0"/>
           <xs:enumeration value="1"/>
           <xs:enumeration value="2"/>
           <xs:enumeration value="3"/>
           <xs:anyAttribute namespace="##any" processContents="lax"/>
         </xs:restriction>
       </xs:simpleContent>
     </xs:complexType>
   </xs:element>
   <xs:element name="AvailableFlag" type="cim:cimBoolean" nillable="true"/>
  <xs:complexType name="EX_DerivedComponent_Type">
    <xs:sequence>
      <xs:element ref="class:AvailableFlag" minOccurs="0"/>
      <xs:element ref="class:EnabledState" minOccurs="0"/>
      <xs:element ref="class:HealthStatus" minOccurs="0"/>
      <xs:element ref="class:InstallDate" minOccurs="0"/>
      <xs:element ref="class:Name"/>
      <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
       <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContent="lax"/>
  </xs:complexType>
  <xs:element name="EX_DerivedComponent" type="class:EX_DerivedComponent_Type"/>
  <xs:element name="RequestStateChange_INPUT">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="RequestedState" nillable="true">
          <xs:complexType>
            <xs:simpleContent>
              <xs:restriction base="cim:cimUnsignedShort">
                <xs:enumeration value="2"/>
                <xs:enumeration value="3"/>
                <xs:enumeration value="4">
                <xs:anyAttribute namespace="##any" processContents="lax"/>
              </xs:restriction>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="TimeoutPeriod" type="cim:cimDateTime" nillable="true"/>
       </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="RequestStateChange_OUTPUT">
     <xs:complexType>
         <xs:sequence>
          <xs:element name="ResultClass" type="cim:cimReference" nillable="true"/>
          <xs:element name="ReturnValue" nillable="true">
    <xs:complexType>
       <xs:simpleContent>
            <xs:restriction base="cim:cimAnySimpleType">
                <xs:simpleType>
                     <xs:union>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedInt">
                                <xs:enumeration value="0"/>
                                <xs:enumeration value="1"/>
                                <xs:enumeration value="2"/>
                            </xs:restriction>
                        </xs:simpleType>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedInt">
                                <xs:minInclusive value="3"/>
                                <xs:maxInclusive value="32767"/>
                            </xs:restriction>
                        </xs:simpleType>
                        <xs:simpleType>
                            <xs:restriction base="xs:unsignedInt">
                                <xs:minInclusive value="32768"/>
                                <xs:maxInclusive value="65535"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:union>
                </xs:simpleType>
                <xs:anyAttribute namespace="##any" processContents="lax"/>
            </xs:restriction>
        </xs:simpleContent>
    </xs:complexType>
          </xs:element>
         </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

C.2.3 EX_AssociationComponent

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
    xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
    xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    …>
  <xs:import
     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
  <xs:element name="AssociatingComponent" type="cim:cimReference"/>
  <xs:element name="AssociatedComponent" type="cim:cimReference"/>
  <xs:element name="WhenAssociated" type="cim:cimDateTime" nillable="true"/>
  <xs:element name="AssocMaintained" type="cim:cimBoolean" nillable="true"/>
  <xs:complexType name="EX_AssociationComponent_Type">
    <xs:sequence>
       <xs:element ref="class:AssociatedComponent"/>
       <xs:element ref="class:AssociatingComponent"/>
       <xs:element ref="class:AssocMaintained" minOccurs="0"/>
       <xs:element ref="class:WhenAssociated" minOccurs="0"/>
       <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContent="lax"/>
  </xs:complexType>
  <xs:element name="EX_AssociationComponent" type="class:EX_AssociationComponent_Type"/>
</xs:schema>
```

C.2.4 Class Hierarchy Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
   xmlns:chier="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import
   namespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/classhiertype.xsd"/>
  <xs:element name="EX_BaseComponent_Class">
    <xs:complexType>
      <xs:complexContent>
        <xs:restriction base="ctype:ClassHierarchyType" />
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
   <xs:element name="EX_DerivedComponent_Class">
     <xs:complexType>
       <xs:complexContent>
         <xs:restriction base="ctype:ClassHierarchyType">
            <xs:sequence>
               <xs:element ref="chier:EX_BaseComponent_Class" />
            </xs:sequence>
         </xs:restriction>
       </xs:complexContent>
     </xs:complexType>
   </xs:element>
  <xs:element name="EX_AssociationComponent_Class">
    <xs:complexType>
      <xs:complexContent>
        <xs:restriction base="ctype:ClassHierarchyType" />
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
<xs:/schema>
```

C.3 WSDL Fragments

This clause contains the WSDL fragments (`wsdl:message`, `wsdl:operation`) that would result from the application of this specification to the EX_DerivedComponent class. This class specifies only one method, `RequestStateChange`.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<wsdl:definitions
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://. . ..wsdl"
   xmlns:cimClass="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
   xmlns:thisWSDL="http://. . ..wsdl"
   …>
  <w:import namespace="http://www.w3.org/2005/08/addressing"
      location="http://www.w3.org/2005/08/addressing/ws-addr.xsd"/>
  <wsdl:types>
    … <!--  Schema of EX_DerivedComponent  -->
  </wsdl:types>
  <wsdl:message name="RequestStateChange_InputMessage">
    <wsdl:part name="body"
      element="cimClass:RequestStateChange_INPUT"/>
  </wsdl:message>
  <wsdl:message name="RequestStateChange_OutputMessage">
    <wsdl:part name="body"
      element="cimClass:RequestStateChange_OUTPUT"/>
  </wsdl:message>
<!--   OPERATION: RequestStateChange
    <wsdl:operation name="RequestStateChange">
      <wsdl:input name="RequestStateChange_InputMessage"
        message="thisWSDL:RequestStateChange_InputMessage"
      <wsdl:output name="RequestStateChange_OutputMessage"
        message="thisWSDL:RequestStateChange_OutputMessage"
    </wsdl:operation>
-->
</wsdl:definitions>
```

C.4 MetaData Fragments

Metadata fragments are generated from the qualifiers that are associated with a class, property, reference, method, or parameter. XML documents that incorporate these fragments must import the cim and cimQ namespaces.

C.4.1 EX_BaseComponent

C.4.1.1 Class Qualifiers

```
<cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
<cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
<cimQ:Description cim:qualifier="true">
   EX_BaseComponent serves as an example base CIM class.
</cimQ:Description>
```

C.4.1.2 Property Qualifiers

C.4.1.2.1 HealthStatus

```
<cimQ:Description cim:qualifier="true">
   A descriptive code of the operational health of a Component.
</cimQ:Description>
<cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
```

C.4.1.2.2 InstallDate

```
<cimQ:Description cim:qualifier="true">
   EX_BaseComponent serves as an example base CIM class.
</cimQ:Description>
<cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
```

C.4.1.2.3 Name

```
<cimQ:Description cim:qualifier="true">
   The Name property defines the label by which the object is known.
</cimQ:Description>
<cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
<cimQ:Required cim:qualifier="true">true</cimQ:Required>
```

C.4.1.2.4 StatusDescriptions

```
<cimQ:Description cim:qualifier="true">
   A set of descriptive statements that can be used to describe the state of an Component.
</cimQ:Description>
<cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>
```

C.4.2 EX_DerivedComponent

C.4.2.1 Class Qualifiers

```
<cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
<cimQ:Description cim:qualifier="true">
   This class extends EX_BaseComponent.
</cimQ:Description>
```

C.4.2.2 Property Qualifiers

C.4.2.2.1 AvailableFlag

```
<cimQ:Description cim:qualifier="true">
   Boolean flag indicating availability of a Component.
```

C.4.2.2.2 EnabledState

```
<cimQ:Description cim:qualifier="true">
   EnabledState is an integer enumeration that indicates the enabled
   and disabled states of a derived Component.
</cimQ:Description>
<cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
<cimQ:Values cim:qualifier="true">Unknown</cimQ:Values>
<cimQ:Values cim:qualifier="true">Other</cimQ:Values>
<cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
<cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
<cim:DefaultValue xsi:type="xs:uint16">3</cim:DefaultValue>
HealthStatus
<cimQ:Description cim:qualifier="true">
   A descriptive code of the operational health of a Component.
</cimQ:Description>
<cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
</cimQ:Description>
```

C.4.2.2.3 InstallDate

```
<cimQ:Description cim:qualifier="true">
   EX_BaseComponent serves as an example base CIM class.
</cimQ:Description>
```

C.4.2.2.4 Name

```
<cimQ:Description cim:qualifier="true">
   The Name property defines the label by which the object is known.
</cimQ:Description>
<cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
<cimQ:Required cim:qualifier="true">true</cimQ:Required>
```

C.4.2.2.5 StatusDescriptions

```
<cimQ:Description cim:qualifier="true">
   A set of descriptive statements that can used to describe the state of an Component.
</cimQ:Description>
<cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>
```

C.4.2.2.6 AvailableFlag

```
<cimQ:Description cim:qualifier="true">
   Boolean flag indicating availability of a Component.
</cimQ:Description>
```

C.4.2.3 Method and Parameter Qualifiers

C.4.2.3.1 RequestStatusChange Method

```
<cimQ:Description cim:qualifier="true">
   Requests that the state of the element be changed to the value
   specified in the RequestedState parameter.
</cimQ:Description>
<cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">..</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">4096</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">4100..32767</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">32768..65535</cimQ:ValueMap>
<cimQ:Values cim:qualifier="true">Completed with No Error</cimQ:Values>
<cimQ:Values cim:qualifier="true">Not Supported</cimQ:Values>
<cimQ:Values cim:qualifier="true">Unknown or Unspecified Error</cimQ:Values>
<cimQ:Values cim:qualifier="true">Failed</cimQ:Values>
<cimQ:Values cim:qualifier="true">DMTF Reserved</cimQ:Values>
<cimQ:Values cim:qualifier="true">Vendor Specific</cimQ:Values>
```

C.4.2.3.2 RequestedState Parameter

```
<cimQ:Description cim:qualifier="true">
   The state requested for the Component.
</cimQ:Description>
<cimQ:In cim:qualifier="true">true</cimQ:In>
<cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
<cimQ:ValueMap cim:qualifier="true">4</cimQ:ValueMap>
<cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
<cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
<cimQ:Values cim:qualifier="true">Shutdown</cimQ:Values>
```

C.4.2.3.3 ResultClass Parameter

```
<cimQ:Description cim:qualifier="true">
   Reference to an instance of some class (undefined in this example)
   that is returned upon completion of the operation.
```

```
</cimQ:Description>
<cimQ:Out cim:qualifier="true">true</cimQ:Out>
<cimQ:In cim:qualifier="true">false</cimQ:In>
```

C.4.2.3.4 TimeoutPeriod Parameter

```
<cimQ:Description cim:qualifier="true">
   A timeout period that specifies the maximum amount of time that the
   client expects the transition to the new state to take.
</cimQ:Description>
<cimQ:In cim:qualifier="true">true</cimQ:In>
```

C.4.3 EX_AssociationComponent

C.4.3.1 Class Qualifiers

```
<cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
<cimQ:Description cim:qualifier="true">
   Indicates that two entites are associated.
</cimQ:Description>
<cimQ:Association cim:qualifier="true">true<cimQ:Association>
```

C.4.3.2 Property Qualifiers

C.4.3.2.1 AssociatedComponent

```
<cimQ:Key cim:qualifier="true">true</cimQ:Key>
<cimQ:Description cim:qualifier="true">
   AssociatedComponent represents another Component (up to 4) that is associated
   with the Component referenced as AssociatingComponent.
</cimQ:Description>
<cimQ:Max cim:qualifier="true">4</cimQ:Max>
```

C.4.3.2.2 AssociatingComponent

```
<cimQ:Key cim:qualifier="true">true</cimQ:Key>
<cimQ:Description cim:qualifier="true">
   An AssociatingComponent represents one Component is associated with the
   component referenced as AssociatedComponent.
</cimQ:Description>
```

C.4.3.2.3 AssocMaintained

```
<cimQ:Description cim:qualifier="true">
  Boolean indicating whether the association is maintained.
</cimQ:Description>
```

C.4.3.2.4 WhenAssociated

```
<cimQ:Description cim:qualifier="true">
  The point in time that the Components were associated.
</cimQ:Description>
```

ANNEX D (Informative)

Collation Optimization Available to Implementers

D.1 Sorting with Limited Character Set

The character set permitted in CIM identifiers is limited to
- U+0030..U+0039 (digits 0-9)
- U+0041..U+005A (alphabetics A-Z)
- U+0061..U+007A (alphabetics a-z)
- U+0052 (underscore)
- U+0080..U+FFEF (the rest of Unicode)

The intention of the preferred collation is that the property name identifier strings should be ordered by a binary sorting of big-endian UCS-2 representation of the characters. For example, ABC, ABc, and AbC sort in this order because

```
ABC 00 41 00 42 00 43
ABc 00 41 00 42 00 63
AbC 00 41 00 62 00 43
```

(and accented As sort in order by Unicode number)

If the identifiers of a MOF do not use the full range of characters, optimization can be applied to the sorting of property identifiers. Specifically, a simple approach exists for the commonly observed case where MOF characters are limited to ASCII-7, i.e., character values < 0x7F.

Most or all MOFs that you will encounter contain only (a subset of) ASCII-7 characters, that is, characters in the range 0x00 to 0x7F. For such MOFs, it is not necessary to cast the property identifier strings into Unicode representation at all. If all the characters are ASCII-7, then the strings can be sorted as simple one-byte sequences.

If the internal representation of character strings (in a CIMOM, protocol adapter, or client application, etc.) is UTF-8, note that ASCII-7 characters are represented unmodified in UTF-8.

Suggested algorithm: Pre-scan the MOF, or at least the set of property identifiers, for characters > 0x7F. If the set contains no such characters > 0x7F, then sort the strings as simple one-byte octet strings.

This case can be used for all currently published DMTF MOFs.

D.2 Note of Caution Concerning Collation

The default Unicode Collation Algorithm orders properties differently from the current DMTF practice. All published CIM XML schemata order properties as described here (sorting by Unicode value). The UCA, by default, uses a different ordering even for the subset of ASCII-7 characters: lower case characters sort before upper case characters. This results in differences in some cases, where, for example, "CPU" and "Caption" appear in one order in the published XSD files but sort in the other order using the UCA.

The intention of the preferred algorithm is to retain current DMTF practice. Properties in XSDs will continue to be in upper-before-lower order. Some existing WS-Man service implementations may be using the default Unicode Collation Algorithm. These implementations will have to become compatible with existing practice.