



1

2

3

4

Document Number: DSP0230

Date: 2008-05-22

Version: 1.0.0

5

WS-CIM Mapping Specification

6

Document Type: Specification

7

Document Status: Final Standard

8

Document Language: E

9

10 Copyright notice

11 Copyright © 2007, 2008 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

12 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
13 management and interoperability. Members and non-members may reproduce DMTF specifications and
14 documents for uses consistent with this purpose, provided that correct attribution is given. As DMTF
15 specifications may be revised from time to time, the particular version and release date should always be
16 noted.

17 Implementation of certain elements of this standard or proposed standard may be subject to third party
18 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
19 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
20 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
21 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
22 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
23 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
24 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
25 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
26 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
27 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
28 implementing the standard from any and all claims of infringement by a patent owner for such
29 implementations.

30

CONTENTS

31	Foreword	5
32	Introduction	6
33	1 Scope	7
34	1.1 In-Scope Features	7
35	1.2 Out-of-Scope Considerations	7
36	2 Conformance	8
37	3 Normative References	8
38	3.1 Approved References	8
39	3.2 Other References	8
40	4 Terms and Definitions	8
41	5 Symbols and Abbreviated Terms	10
42	6 Namespace Prefixes and Schema Locations	11
43	7 Dereferencing Schema URI Locations in Order to Access XML Schema	12
44	8 Mapping Primitive Datatypes	13
45	8.1 cimDateTime Datatype	14
46	8.2 CIM References	15
47	8.3 cimAnySimpleType Datatype	16
48	8.4 Derivation by Restriction	17
49	8.5 WS-CIM Canonical Values	17
50	9 CIM Class to XML Schema Mappings	17
51	9.1 Class Namespace	17
52	9.2 Properties	18
53	9.3 Class Structure	27
54	9.4 Class Inheritance	29
55	9.5 Method Parameters	30
56	9.6 CIM Instances	32
57	9.7 Superclass Class Representation	33
58	10 CIM Methods to WSDL Mappings	34
59	10.1 Defining WSDL Message Structures	34
60	10.2 Defining WSDL Operation Structures	35
61	10.3 Defining wsa:Actions	35
62	11 Qualifier Mappings	36
63	11.1 General Format	36
64	11.2 Mapping CIM Qualifiers to XSD Elements	39
65	11.3 Inheritance of Qualifiers	41
66	ANNEX A (Informative) Schemas	43
67	A.1 Common WS-CIM Schema: DSP8004	43
68	A.2 Qualifiers Schema: DSP8005	46
69	A.3 Class Hierarchy Type Schema: DSP8006	48
70	ANNEX B (Informative) Conventions	49
71	ANNEX C (Informative) Examples	50
72	C.1 MOF Definitions	50
73	C.2 XSD	51
74	C.3 WSDL Fragments	55
75	C.4 MetaData Fragments	56
76		

77 **Tables**

78 Table 1 – Namespaces 11

79 Table 2 – Namespace Prefixes 11

80 Table 3 – Schema URI Locations 12

81 Table 4 – XSD DSP Numbers..... 12

82 Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes 13

83 Table 6 – Rules for Converting datetime to cimDateTime 15

84 Table 7 – Rules for Converting cimDateTime to datetime 15

85 Table 8 – CIM Qualifiers Mapped to XSD Elements..... 39

86 Table 9 – Rules of Qualifier Inheritance..... 41

87

88

Foreword

89 The *WS-CIM Mapping Specification* (DSP0230) was prepared by the DMTF WBEM Infrastructure and
90 Protocols Working Group.

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

93 Authors:

- 94 • Akhil Arora, Sun Microsystems, Inc.
- 95 • Ed Boden, IBM
- 96 • Mark Carlson, Sun Microsystems, Inc. (past Co-Chair)
- 97 • Josh Cohen, Microsoft Corporation
- 98 • Asad Faizi, Microsoft Corporation (past Editor)
- 99 • Vincent Kowalski, BMC Software, Inc. (past Co-Chair)
- 100 • Heather Kreger, IBM
- 101 • Richard Landau, Dell Inc.
- 102 • Tom Maguire, EMC
- 103 • Andreas Maier, IBM
- 104 • James Martin, Intel Corporation
- 105 • Bryan Murray, Hewlett-Packard
- 106 • Brian Reistad, Microsoft Corporation
- 107 • Mitsunori Satomi, Hitachi
- 108 • Sharon Smith, Intel Corporation
- 109 • Kirk Wilson, CA, Inc. (past Editor)
- 110 • Dr. Jerry Xie, Intel Corporation
- 111 • Steve Hand, Symantec Corporation (Editor and Chair)

112

113

114

115

Introduction

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

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

128

WS-CIM Mapping Specification

129 1 Scope

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

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

139 1.1 In-Scope Features

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

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

147 1.2 Out-of-Scope Considerations

148 The following items are outside the scope of this specification:

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

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

172 2 Conformance

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

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

177 3 Normative References

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

181 3.1 Approved References

- 182 DMTF [DSP0004](#), *Common Information Model (CIM) Infrastructure Specification*, version 2.3
183 DMTF [DSP4009](#), *Process for publishing XML schema, XML documents and XSLT stylesheets*
184 IETF [RFC 3987](#), *Internationalized Resource Identifiers (IRIs)*, January 2005
185 IETF [RFC 3986](#), *Uniform Resource Identifier (URI): Generic Syntax*, January 2005
186 W3C, [Namespaces in XML](#), W3C Recommendation, 14 January 1999. (This version of the *XML*
187 *Information Set Recommendation* is available at <http://www.w3.org/TR/1999/REC-xml-names-19990114>.
188 The latest version of *Namespaces in XML* is available at <http://www.w3.org/TR/REC-xml-names>.)
189 W3C, [Web Services Addressing \(WS-Addressing\)](#), W3C Member Submission, 10 August 2004
190 W3C, [Web Services Addressing \(WS-Addressing\) 1.0 – Core](#), W3C Recommendation, 9 May 2006
191 W3C, [Web Services Description Language \(WSDL\) 1.1](#), W3C Note, 15 March 2001
192 W3C, [XML Schema Part 2: Datatypes](#), W3C Recommendation, October 2004
193 W3C, [XML Schema Part 1: Structures](#), W3C Recommendation, October 2004

194 3.2 Other References

- 195 ISO, [ISO/IEC Directives, Part 2](#), *Rules for the structure and drafting of International Standards*

196 4 Terms and Definitions

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

198 4.1

199 **can**

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

- 201 **4.2**
202 **cannot**
203 used for statements of possibility and capability, whether material, physical or causal
- 204 **4.3**
205 **conditional**
206 indicates requirements to be followed strictly in order to conform to the document when the specified
207 conditions are met
- 208 **4.4**
209 **mandatory**
210 indicates requirements to be followed strictly in order to conform to the document and from which no
211 deviation is permitted
- 212 **4.5**
213 **may**
214 indicates a course of action permissible within the limits of the document
- 215 **4.6**
216 **need not**
217 indicates a course of action permissible within the limits of the document
- 218 **4.7**
219 **optional**
220 indicates a course of action permissible within the limits of the document
- 221 **4.8**
222 **shall**
223 indicates requirements to be followed strictly in order to conform to the document and from which no
224 deviation is permitted
- 225 **4.9**
226 **shall not**
227 indicates requirements to be followed strictly in order to conform to the document and from which no
228 deviation is permitted
- 229 **4.10**
230 **should**
231 indicates that among several possibilities, one is recommended as particularly suitable, without
232 mentioning or excluding others, or that a certain course of action is preferred but not necessarily required
- 233 **4.11**
234 **should not**
235 indicates that a certain possibility or course of action is deprecated but not prohibited
- 236 **4.12**
237 **Global Element Declaration**
238 element declaration in an XML Schema that places the element as an immediate child of the root element
239 of the schema
- 240 **4.13**
241 **Managed Object Format**
242 an IDL based language, defined by the DMTF, expressing the structure, behavior, and semantics of a
243 CIM class and its instances.

- 244 **4.14**
245 Uniform Record Identifier
246 a Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or
247 physical resource. See [RFC3986](#).
- 248 **4.15**
249 **Runtime**
250 describes the situation where XML instances are produced that are conformant to this specification.
- 251 **4.16**
252 **WS-CIM**
253 of or pertaining to this specification.
254 NOTE: "WS-CIM" is not an acronym; it should be treated simply as the name of the contents of the specification.
- 255 **4.17**
256 **XML instance document**
257 an XML document that conforms to a specified XML Schema
258 As used in this specification, *XML instance document* refers to a document that conforms to an XML
259 Schema that conforms to the rules in clause 9.
- 260 **4.18**
261 **XSDL**
262 offers facilities for describing the structure and constraining the contents of XML documents, including
263 those which exploit the XML Namespace facility. XSDL documents have the '.xsd' file extension. See:
264 <http://www.w3.org/TR/xmlschema11-1/>.

265 **5 Symbols and Abbreviated Terms**

266 The following symbols and abbreviations are used in this document.

- 267 **5.1**
268 **GED**
269 Global Element Declaration
- 270 **5.2**
271 **MOF**
272 Managed Object Format
- 273 **5.3**
274 **URI**
275 Uniform Resource Identifier
- 276 **5.4**
277 **WSDL**
278 Web Services Description Language
- 279 **5.5**
280 **XML**
281 Extensible Mark-up Language
- 282 **5.6**
283 **XSDL**
284 XML Schema Definition Language

285 6 Namespace Prefixes and Schema Locations

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

288 This specification defines namespaces as shown in Table 1.

289 **Table 1 – Namespaces**

Namespace	Description
http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/ClassName	Contains the schema for the class <i>ClassName</i>
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

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

292 **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://www.w3.org/2004/08/addressing (see WS-Addressing)
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

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

294 **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

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

296 **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/cim-schema/Y/classhierarchy.xsd	DSP8006

297 7 Dereferencing Schema URI Locations in Order to Access XML 298 Schema

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

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

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

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

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

311 EXAMPLE:

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

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

318 EXAMPLE:

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

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

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

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

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

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

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

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

331 **7.1.1 Validation of CIM Instances**

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

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

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

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

349 **8 Mapping Primitive Datatypes**

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

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

355 **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

CIM Datatype	Corresponding Base XSD Datatypes	WS-CIM Datatypes
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.

356 NOTE: For mapping of array properties, see 9.2.2.

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

- 359 Octetstring
- 360 EmbeddedInstance
- 361 EmbeddedObject

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

364 8.1 cimDateTime Datatype

365 The cim:cimDateTime datatype is defined as follows:

```

366 <xs:complexType name="cimDateTime">
367   <xs:choice>
368     <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
369     <xs:element name="Interval" type="xs:duration"/>
370     <xs:element name="Date" type="xs:date"/>
371     <xs:element name="Time" type="xs:time"/>
372     <xs:element name="Datetime" type="xs:dateTime"/>
373   </xs:choice>
374 <xs:anyAttribute namespace="##any" processContents="lax"/>
375 </xs:complexType>

```

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

377 **Table 6 – Rules for Converting `datetime` to `cimDateTime`**

CIM <code>datetime</code> Use Case	String Condition	<code>cim:cimDateTime</code> 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 <code>hhmmss.mmmmmm</code> portion, and only asterisks in the <code>yyyymmdd</code> portion	Time
date	String contains "+" or "-" and no asterisks in the <code>yyyymmdd</code> portion, and only asterisks in the <code>hhmmss.mmmmmm</code> portion	Date
Other	String asterisks other than indicated above	<code>CIM_DateTime</code>

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

380 **Table 7 – Rules for Converting `cimDateTime` to `datetime`**

<code>cim:cimDateTime</code> Element	Representation of <code>datetime</code> in CIM
Interval	CIM <code>datetime</code> 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 <code>0000000223****.*****:000</code>
Datetime	CIM <code>datetime</code> 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 <code>datetime</code> string does not contain any asterisks, because this XML element is used only when the original CIM <code>datetime</code> satisfies this condition.
Time	CIM <code>datetime</code> 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 <code>datetime</code> string does not contain any asterisks in the <code>hhmmss.mmmmmm</code> portion, and contains only asterisks in the <code>yyyymmdd</code> portion, because this XML element is used only when the original CIM <code>datetime</code> satisfies this condition.
Date	CIM <code>datetime</code> 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 <code>datetime</code> string does not contain any asterisks in the <code>yyyymmdd</code> portion, and contains only asterisks in the <code>hhmmss.mmmmmm</code> portion, because this XML element is used only when the original CIM <code>datetime</code> satisfies this condition.
<code>CIM_DateTime</code>	CIM <code>datetime</code> with a string equal to the XML element text.

381 **8.2 CIM References**

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

```

383 <xs:complexType name="cimReference">
384   <xs:sequence>
385     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
386   </xs:sequence>
387 <xs:anyAttribute namespace="##any" processContents="lax"/>
388 </xs:complexType>
    
```

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

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

401 EXAMPLE: An example of the use of WS-Addressing schemas as a transport reference, mapped to the
402 `AssociatedComponent` property, is as follows:

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

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

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

```
410  
411 <AssociatedComponent  
412   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">  
413   <wsa:Address> . . . </wsa:Address>  
414   . . . <!-- Other EPR elements as defined in the 2004/08 specification -->  
415 </AssociatedComponent>
```

416 8.3 `cimAnySimpleType` Datatype

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

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

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

436 8.4 Derivation by Restriction

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

```
439 . . .
440 <this:Name xmlns:AdditionalAttribute=". . .">
441     myName
442 </this:Name>
443 . . .
```

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

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

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

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

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

455 8.5 WS-CIM Canonical Values

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

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

462 9 CIM Class to XML Schema Mappings

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

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

468 9.1 Class Namespace

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

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

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

473 ['http://schemas.dmtf.org/wbem/wscim/'](http://schemas.dmtf.org/wbem/wscim/) wscim-major-version '/cim-schema' cim-schema-major-
474 version '/' cim-class-schema '_' cim-class-name

475 Where:

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

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

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

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

- 484 • The process and rules for the publication of the schema documents that define class
485 namespaces are defined in [DSP4009](#).

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

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

489 9.2 Properties

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

493 9.2.1 General Principles

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

496 9.2.1.1 General Rules

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

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

512 nillable is false), if the CIM property has a *Key* or *Required* qualifier with an effective
 513 value of *true*. (See 11.3 for rules regarding the inheritance of qualifiers.)

514 NOTE: These rules do not apply to array properties. All GEDs that represent array properties shall be
 515 specified with *nillable="true"*.

- 516 • The CIM Schema may assign default initializer values directly to properties, as, for example, in
 517 the MOF construct `uint16 EnabledState=3` (see [DSP0004](#)). Default initializer values shall
 518 be mapped to a metadata fragment using `<cim:DefaultValue>`. Default initializer values
 519 shall not be mapped to the `xs:default` attribute, which carries different semantics than CIM
 520 Schema default values.

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

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

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

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

```
536 class EX_BaseComponent {
537     datetime InstallDate;
538     [ ...
539         Required, MaxLen ( 1024 ) ]
540     string Name;
541     string StatusDescriptions[];
542     string HealthStatus;
543 };
```

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

```
546 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
547 <xs:element name="Name">
548   <xs:complexType>
549     <xs:simpleContent>
550       <xs:restriction base="cim:cimString">
551         <xs:maxLength value="1024"/>
552         <xs:anyAttribute namespace="##any" processContents="lax"/>
553       </xs:restriction>
554     </xs:simpleContent>
555   </xs:complexType>
556 </xs:element>
```

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

559 9.2.1.2 Runtime Rules for Attribute Value Assignment

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

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

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

```
570 class EX_SomeClass {
571     ...
572     [... Key]
573     string Name;
574 }
```

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

```
576 <EX_SomeClass>
577     ...
578     <Name cim:Key="true">MyName</Name>
579     ...
580 </EX_SomeClass>
```

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

582 9.2.2 Array Properties

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

584 9.2.2.1 General Rules

585 The rule for representing arrays is as follows:

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

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

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

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

```
595 [...
596     ArrayType ( "Indexed" ) ]
597 string StatusDescriptions[];
```

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

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

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

```

602 <EX_BaseComponent>
603   . . .
604   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
605   <StatusDescriptions>AnotherStatusDescription</StatusDescriptions>
606   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
607   . . .
608 </EX_BaseComponent>

```

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

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

```

612 [ . . .
613   MaxLen ( 64 ) ]
614 string StatusDescriptions[];

```

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

```

618 <xs:element name="StatusDescriptions nillable="true">
619   <xs:complexType>
620     <xs:restriction base="cim:cimString">
621       <xs:maxLength value="64"/>
622       <xs:anyAttribute namespace="##any" processContents="lax"/>
623     </xs:restriction>
624   </xs:complexType>
625 </xs:element>

```

626 9.2.2.2 Runtime Rules for Arrays

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

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

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

```

637 <EX_BaseComponent>
638   . . .
639   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
640   <StatusDescriptions xsi:nil="true"/>
641   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
642   . . .
643 </EX_BaseComponent>

```

644 9.2.3 Properties with a ValueMap Qualifier

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

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

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

- 651 • If the `ValueMap` consists of only discrete values, the `ValueMap` shall be mapped to a
652 `<xs:restriction>` consisting of an enumeration as follows:
 - 653 – The base type of the restriction shall be the WS-CIM datatype corresponding to the CIM
654 datatype of the property (see 8).
 - 655 – Each discrete value of the `ValueMap` shall be mapped to a corresponding
656 `<xs:enumeration>` element within the restriction.
- 657 • If the `ValueMap` contains a value specifying a range, the whole `ValueMap` shall be mapped to a
658 `<xs:restriction>` consisting of a `<xs:union>` as follows:
 - 659 – The base type of the restriction shall be `cim:cimAnySimpleType` (see 8.3).
 - 660 – The elements of the `<xs:union>` shall be determined according to the following rules:
 - 661 • Discrete values shall be mapped to elements to an `<xs:restriction>` as
662 described in the first rule with the exception that the base type of the restriction shall
663 be the corresponding base XSD type of the CIM datatype of the property (see
664 clause 8);
 - 665 • Bounded ranges (*m..n*) shall be mapped to an `<xs:restriction>` consisting of
666 `<xs:minInclusive="m">` and `<xs:maxInclusive="n">` where the restriction
667 base type shall be the corresponding base XSD type of the CIM datatype of the
668 property;
 - 669 • Unbounded ranges open on the left (*...n*) shall be mapped to an
670 `<xs:restriction>` consisting of `<xs:maxInclusive="n">` where the restriction
671 base type shall be the corresponding base XSD type of the CIM datatype of the
672 property;
 - 673 • Unbounded ranges open on the right (*m...*) shall be mapped to an
674 `<xs:restriction>` consisting of `<xs:minInclusive="m">` where the restriction
675 base type shall be the corresponding base XSD type of the CIM datatype of the
676 property;
 - 677 • Open ranges (*..*) shall be mapped to an `<xs:union>` consisting of all discrete values
678 and/or ranges that are unclaimed by the other values and ranges in the `ValueMap` by
679 applying the preceding rules for constructing the elements of the `<xs:union>`
680 recursively.

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

```
682 [ ...
683 ValueMap { "OK", "Error", "Unknown" } ]
684 string HealthStatus;
```

685 The `HealthStatus` property is therefore mapped as follows:

```
686 <xs:element name="HealthStatus" nillable="true">
687   <xs:complexType>
688     <xs:simpleContent>
689       <xs:restriction base="cim:cimString">
690         <xs:enumeration value="OK"/>
691         <xs:enumeration value="Error"/>
692         <xs:enumeration value="Unknown"/>
693         <xs:anyAttribute namespace="##any" processContents="lax"/>
694       </xs:restriction>
```

```

695     </xs:simpleContent
696   </xs:complexType>
697 </xs:element>

```

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

```

699 [...]
700 ValueMap { "0", "1", "2", "3..15999", "16000..65535" },
701   Values { "Unknown", "Other", "Not Applicable", "DMTF Reserved",
702     "Vendor Reserved" }]
703 uint16 PortType;

```

704 The PortType property is therefore mapped as follows:

```

705 <xs:element name="PortType" nillable="true">
706   <xs:complexType>
707     <xs:simpleContent>
708       <xs:restriction base="cim:cimAnySimpleType">
709         <xs:simpleType>
710           <xs:union>
711             <xs:simpleType>
712               <xs:restriction base="xs:unsignedShort">
713                 <xs:enumeration value="0"/>
714                 <xs:enumeration value="1"/>
715                 <xs:enumeration value="2"/>
716               </xs:restriction>
717             </xs:simpleType>
718             <xs:simpleType>
719               <xs:restriction base="xs:unsignedShort">
720                 <xs:minInclusive value="3"/>
721                 <xs:maxInclusive value="15999"/>
722               </xs:restriction>
723             </xs:simpleType>
724             <xs:simpleType>
725               <xs:restriction base="xs:unsignedShort">
726                 <xs:minInclusive value="16000"/>
727                 <xs:maxInclusive value="65535"/>
728               </xs:restriction>
729             </xs:simpleType>
730           </xs:union>
731         </xs:simpleType>
732         <xs:anyAttribute namespace="##any" processContents="lax"/>
733       </xs:restriction>
734     </xs:simpleContent>
735   </xs:complexType>
736 </xs:element>

```

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

```

739 [...]
740 ValueMap { "1", "2", "3", "4", "5", "6", "7", "..", "16000.." },
741   Values { "Other", "Create", "Delete", "Detect", "Read", "Write",
742     "Execute", "DMTF Reserved", "Vendor Reserved" }]
743 uint16 Activities;

```

744 The Activities property is therefore mapped as follows:

```

745 <xs:element name="Activities" nillable="true">
746   <xs:complexType>
747     <xs:simpleContent>
748       <xs:restriction base="cim:cimAnySimpleType">
749         <xs:simpleType>
750           <xs:union>
751             <xs:simpleType>
752               <xs:restriction base="xs:unsignedShort">
753                 <xs:enumeration value="1"/>
754                 <xs:enumeration value="2"/>
755                 <xs:enumeration value="3"/>
756                 <xs:enumeration value="4"/>
757                 <xs:enumeration value="5"/>
758                 <xs:enumeration value="6"/>
759                 <xs:enumeration value="7"/>
760               </xs:restriction>
761             </xs:simpleType>
762             <xs:simpleType>
763               <xs:union>
764                 <xs:simpleType>
765                   <xs:restriction base="xs:unsignedShort">
766                     <xs:enumeration value="0"/>
767                   </xs:restriction>
768                 </xs:simpleType>
769                 <xs:simpleType>
770                   <xs:restriction base="xs:unsignedShort">
771                     <xs:minInclusive value="8"/>
772                     <xs:maxInclusive value="15999"/>
773                   </xs:restriction>
774                 </xs:simpleType>
775               </xs:union>
776             </xs:simpleType>
777             <xs:simpleType>
778               <xs:restriction base="xs:unsignedShort">
779                 <xs:minInclusive value="16000"/>
780               </xs:restriction>
781             </xs:simpleType>
782           </xs:union>
783         </xs:simpleType>
784         <xs:anyAttribute namespace="##any" processContents="lax"/>
785       </xs:restriction>
786     </xs:simpleContent>
787   </xs:complexType>
788 </xs:element>

```

789 9.2.4 Octetstring Properties

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

793 **9.2.4.1 General Rules**

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

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

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

```
801 [ ...
802     Description ("The DER-encoded raw public key. " ),
803     OctetString ]
804 uint8 PublicKey[];
```

805 It would be represented by the following XSD:

806

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

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

808

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

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

```
810 [...
811     Description ("A CRL, or CertificateRevocationList, is a list of certificates which the "
812                 "CertificateAuthority has revoked and which are not yet expired. " ),
813     Octetstring]
814 string CRL[];
```

815 It would be represented by the following XSD:

816

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

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

```
818 <CRL>0x000000F976H8A4...</CRL>
819 <CRL>0x000000C675D4G1...</CRL>
820 <CRL>0x000000D8B1H335...</CRL>
```

821 **9.2.4.2 Runtime Value Conversion Rules**822 This clause defines the normative rules for the runtime conversion rules for values of octetstring
823 properties.824 The hex format for the `string` array variant of octetstrings is used to avoid additional conversion steps in
825 the XML protocol layer, which would need to convert the hex encoding generated by the CIM provider to
826 binary and then convert that binary to base64. The values in the preceding examples (see 9.2.4.1) are
827 obtained by applying the following runtime value conversion rules:

- 828 • A `uint8` array that is designated as an octetstring shall be converted to its corresponding
829 representation in `base64Binary` such that the ordered set of array elements is concatenated into
830 a binary multi-octet string, which is converted to base64 encoding. This encoding represents the
831 `base64Binary` value. The order of the unsigned 8-bit integer array shall be preserved when
832 mapped to the characters of the XML value.
- 833 • A `string` array that is designated as an octetstring shall be converted to its corresponding
834 representation in `hexBinary` such that for each string array element, one `hexBinary` element is
835 created, with the unchanged value of the string array element.

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

838 9.2.5 EmbeddedObject and EmbeddedInstance Properties

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

843 9.2.5.1 General Rules

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

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

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

```
850 [ ...  
851     EmbeddedObject ( "... " ) ]  
852 string TheObject;
```

853 It would have the following XSD representation:

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

855 9.2.5.2 Runtime Value Conversion Rules

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

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

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

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

```
871 class CIM_Part {  
872     string Label;  
873     int PartNo;  
874 };  
875 class CIM_Component {  
876     [Key]  
877     string ID;  
878     [EmbeddedInstance("CIM_Part")]  
879     string Part;  
880 };
```

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

```
883 <CIM_Component xmlns="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Component"
884   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
885   <ID>ual23</ID>
886   <Part xmlns:e="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Part"
887     xsi:type="e:CIM_Part_Type">
888     <e:Label>Front Panel</e:Label>
889     <e:PartNo>19932</e:PartNo>
890   </Part>
891 </CIM_Component>
```

892 9.3 Class Structure

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

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

897 9.3.1 General Rules

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

- 899 • The structure of the CIM class shall be mapped to an XML global complex type definition. The
 900 definition of this complex type shall comply with the following rules:
 - 901 – The name of this type shall match that of the CIM class name, including the CIM schema
 902 name, with the suffix `_Type`.

- 903 – The complex type shall consist of an `<xs:sequence>` that contains the set of elements
 904 referring to the GEDs that define the properties of the class (see 9.2). These elements
 905 have the following form:

```
906 '<xs:element ref=' QName ' . ' Attributes '>'
```

907 Where:

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

- 910 – Elements that belong to the class complex type shall be alphabetically ordered with the
 911 type definition based on their CIM property name. The collation sequence shall be
 912 according to the character set defined in [DSP0004](#).

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

- 914 • All elements that do not represent array properties shall have `minOccurs="0"` except for
 915 elements that correspond to properties that are designated with a Key or Required qualifier
 916 with an effective value of `true`. Elements that do not represent arrays and represent key and
 917 required properties shall have `minOccurs="1"`. Because 1 is the default value for
 918 `minOccurs`, it does not need to be explicitly expressed.
- 919 • All elements that represent array properties shall have `minOccurs="0"`. If the array size is
 920 specified in the CIM definition, the array property shall have `maxOccurs="array size"`. If
 921 the array size is not specified, the array property shall have `maxOccurs="unbounded"`.
- 922 • All elements except arrays shall have `maxOccurs="1"`. Because 1 is the default value for
 923 `maxOccurs`, it does not need to be explicitly expressed.

- 924
- 925
- 926
- 927
- 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"`.
- 928
- 929
- 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:


```
930
931 <xs:any namespace="##other" processContents="lax" minOccurs="0"
932     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:


```
933
934
935 <xs:anyAttribute namespace="##any" processContent="lax" />
```
- 936
- 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.

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

```
940 <xs:complexType name="EX_BaseComponent_Type">
941   <xs:sequence>
942     <xs:element ref="class:HealthStatus" minOccurs="0" />
943     <xs:element ref="class:InstallDate" minOccurs="0" />
944     <xs:element ref="class:Name" />
945     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded" />
946     <xs:any namespace="##other" processContents="lax" minOccurs="0"
947         maxOccurs="unbounded" />
948   </xs:sequence>
949   <xs:anyAttribute namespace="##any" processContent="lax" />
950 </xs:complexType>
951 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type" />
```

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

953 9.3.2 Runtime Property Value Rules

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

- 956
- 957
- 958
- 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.
- 964
- 965
- 966
- 967
- 968
- 969

970 • If a `Version` qualifier is associated with the CIM class, the `cim:Version` attribute may be
 971 applied to the element that represents the class in an XML instance document. Use of the
 972 `cim:Version` attribute shall conform to the following rule:

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

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

```
978     [... Version ( "2.10.2" ) ]
979     class EX_SomeClass {
980     ...
981     }
```

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

```
983     <EX_SomeClass cim:Version="2.10.2">
984     ...
985     </EX_SomeClass>
```

986 9.4 Class Inheritance

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

988 Class inheritance is governed by the following rules:

- 989 • Besides including the GEDs for the properties defined in a class (see 9.2.1.1), the namespace
 990 for a class shall also include the GEDs for properties inherited from its superclasses. The class
 991 type definition shall contain references to GEDs for all properties defined in and inherited into
 992 the class according the rules in 9.3.1.
- 993 • In general, classes inherit all properties specified in or inherited by their superclasses along with
 994 all qualifiers that are specified as `ToSubClass`. However, properties with the same name may
 995 be encountered within an inheritance chain. The `Override` qualifier determines special
 996 behaviors that shall be observed by conversion algorithms when encountering properties with
 997 duplicate names in the inheritance chain. The following rules govern the mapping of properties
 998 with duplicate names:
 - 999 – When multiple properties in the inheritance chain that have the same name are not
 1000 overridden (that is, the effective value of the `Override` qualifier throughout the inheritance
 1001 chain is `NULL`), the property and its qualifiers in the most derived class shall be mapped.
 1002 All other duplicate named properties shall not be mapped.
 - 1003 – When a property in a derived class overrides another property (of the same name and
 1004 type) in a superclass, the property in the most derived class shall be mapped, including all
 1005 qualifiers inherited from the overridden property. The overridden property shall not be
 1006 mapped.
- 1007 • The inheritance of qualifiers that pertain to properties shall comply with the inheritance rules
 1008 regarding qualifiers in C.3.
- 1009 • The definition of a derived class shall follow all other rules for constructing classes as defined in
 1010 9.3.1.

1011 NOTE: The metadata fragments for a property shall include any inherited qualifiers, subject to the qualifier inheritance
 1012 rules defined in 11.3. For more information about metadata fragments, see clause 11.

1013 Inheritance rests on the same type of examples presented in 9.2 and 9.3. The only addition is that the
1014 properties inherited from a class's superclasses are included in the GEDs and class complex type
1015 definition. For a complete example of inheritance, see C.2.2.

1016 9.5 Method Parameters

1017 The invocation of a CIM extrinsic method is represented by two separate messages:

- 1018 • the request input message, which represents the invocation of the method and the set of input
1019 parameters
- 1020 • the response output message, which represents the output parameters and the method return
1021 value in the successful case

1022 This clause specifies the XML Schema for these elements. These elements are then included as parts in
1023 the WSDL input and output messages (see 10.1).

1024 9.5.1 General Rules

1025 The rules in this clause apply to mapping method input and output parameters and method return values.

1026 The GED used for the request input message is called the *input message GED*. The GED used for the
1027 response output message is called the *output message GED*. The following rules specify the definition of
1028 these GEDs:

- 1029 • The class namespace of the CIM class being mapped shall contain the input and output
1030 message GEDs of all methods owned by the class and inherited from the superclasses. See
1031 9.5.2, which defines class ownership of methods inherited from superclasses.
- 1032 • The names of these GEDs are defined by the following rules:
 - 1033 – The `name` of the input message GED shall be the name of the CIM method with `_INPUT`
1034 appended.
 - 1035 – The `name` of the output message GED shall be the name of the CIM method with `_OUTPUT`
1036 appended.
- 1037 • Each GED shall be defined as a complex type containing an `xs:sequence` of in-line elements
1038 that represent the respective input or output parameters and return value of the CIM method as
1039 immediate children. This structure is further defined in the rest of this clause.

1040 The following rules define the mapping of individual input and output parameters and the return value of
1041 the CIM method, and the structure of the complex type used for defining the GEDs:

- 1042 • Input and output parameters of a CIM method shall be mapped to elements with the same
1043 name as the corresponding parameter name. The following rules define the features of these
1044 elements:
 - 1045 – The type of an element that represents an input or output parameter shall be mapped as
1046 defined in clause 8.
 - 1047 – Parameters that are not qualified with a `Required` qualifier shall be mapped to elements
1048 that contain the `nillable="true"` attribute.
- 1049 • The complex type used to define the type of the input message GED shall contain all and only
1050 those elements that correspond to method parameters that have their `In` qualifier effectively
1051 defined as `true`. The sequence of these elements in the complex type shall correspond to the
1052 sequence of the input parameters in the CIM definition of the method.

1053 If the method has no input parameters, the complex type used in the GED shall be empty (that
1054 is, `<xs:complexType/>`).

- 1055 • The complex type used to define the type of the output message GED shall contain all elements
1056 that correspond to method parameters that have their `Out` qualifier effectively defined as `true`.
1057 The sequence of these elements in the complex type shall correspond to the sequence of the
1058 output parameters in the CIM definition of the method.
- 1059 • The complex type used to define the output message GED shall contain an element of
1060 name="ReturnValue" as the final element in its sequence. The following rules govern the
1061 structure of this element:
- 1062 – The XSD type of this element shall be mapped from the CIM method type in compliance
1063 with the datatype conversion defined in clause 8.
- 1064 – The element shall include the attribute `nillable="true"`. (See the following note.)
- 1065 • Parameters and return values may be defined in CIM with `ValueMap` qualifiers. Mapping these
1066 `ValueMaps` to metadata fragments is required (see 11.1). In addition, a `ValueMap` shall be
1067 mapped to an enumeration/union associated with the mapped parameter or return value in the
1068 XSD (see 11.2). The mapping shall conform to the rules for mapping `ValueMaps` described in
1069 9.2.3.
- 1070 Notes on the preceding rules:
- 1071 • A parameter shall occur in both the complex types used to define the input and output message
1072 GEDs if it has both the `In` and `Out` qualifiers effectively defined as `true`.

1073 [DSP0004](#) allows NULL as the default return value for all methods. Thus, this specification must allow for
1074 the possibility that the return value of any method may be NULL.

1075 9.5.2 Inheritance of Methods

1076 Classes may inherit some of the methods that they own. In general, classes inherit all methods specified
1077 in or inherited by their superclasses, along with all qualifiers that are specified as `ToSubClass`. The
1078 `Override` qualifier, however, determines special behavioral considerations on the part of conversion
1079 algorithms. The following rules govern the inheritance of methods under conditions of override:

- 1080 • When multiple methods in the inheritance chain that have the same name are not overridden,
1081 the method in the most derived class shall be mapped. Any other duplicate, but not overridden,
1082 methods shall not be mapped.
- 1083 • When a method in a derived class overrides another method (of the same name and signature)
1084 in a superclass, the method in the most derived class shall be mapped, including all qualifiers
1085 inherited from the overridden methods. The overridden method shall not be mapped.
- 1086 • The inheritance of qualifiers pertaining to methods shall comply with the inheritance rules
1087 regarding qualifiers in C.3.

1088 EXAMPLE: As an example of the preceding rules, consider the following MOF method definition extracted from the
1089 example in C.1.2. (See C.2.2 for the complete example.)

```
1090 class EX_DerivedComponent
1091 {
1092   ...
1093   uint32 RequestStateChange([IN] uint16 RequestedState, [OUT] [IN(False)] CIM_SomeClass REF
1094   ResultClass, [IN] datetime TimeoutPeriod);
1095 };
```

1096 The input parameters, designated in the MOF by the `In` qualifier, would be mapped as follows:

```
1097 <xs:element name="RequestStateChange_INPUT">
1098 <xs:complexType>
1099   <xs:sequence>
```

```

1100     <xs:element name="RequestedState" type="cim:cimUnsignedShort"
1101         nillable="true"/>
1102     <xs:element name="TimeoutPeriod" type="cim:cimDateTime"
1103         nillable="true"/>
1104     </xs:sequence>
1105 </xs:complexType>
1106 </xs:element>

```

1107 The output parameters, designated in the MOF by the `Out` qualifier, would be mapped in the following
 1108 way. Note that the complex type includes an element that represents the return value of the CIM method.

```

1109 <xs:element name="RequestStateChange_OUTPUT">
1110 <xs:complexType>
1111   <xs:sequence>
1112     <xs:element name="ResultClass" type="cim:cimReference"
1113         nillable="true"/>
1114     <xs:element name="ReturnValue" type="cim:cimUnsignedInt"
1115         nillable="true"/>
1116   </xs:sequence>
1117 </xs:complexType>
1118 </xs:element>

```

1119 9.5.3 Parameters and Return Values with ValueMaps

1120 Input and output parameters and return values of the method may be specified with a `ValueMap` qualifier.
 1121 The `ValueMap` shall be mapped to the XSD element that represents the parameter or return value.

1122 The `RequestedState` input parameter must be defined as an enumeration in accordance with its
 1123 `ValueMap` (see C.1.2 for this `ValueMap`):

```

1124 <xs:element name="RequestedState" nillable="true">
1125   <xs:complexType>
1126     <xs:simpleContent>
1127       <xs:restriction base="cim:cimUnsignedShort">
1128         <xs:enumeration value="2"/>
1129         <xs:enumeration value="3"/>
1130         <xs:enumeration value="4"/>
1131         <xs:anyAttribute namespace="##any" processContents="lax"/>
1132       </xs:restriction>
1133     </xs:simpleContent>
1134   </xs:complexType>
1135 </xs:element>

```

1136 9.6 CIM Instances

1137 This clause describes the representation of CIM instances. The intended scope is all representations of
 1138 CIM instances used in any protocols.

1139 CIM instances are represented according to the following rules:

- 1140 • Representations of CIM instances shall be XML instance documents that conform to the XSD
 1141 schema for their CIM creation class, as defined in clause 9.
- 1142 • The class namespace used within an instance document shall be a namespace URI and it shall
 1143 be defined as follows:

1144 [http://schemas.dmtf.org/wbem/wscim/ wscim-major-version /cim-schema/ cim-schema-major-
 version /' cim-class-schema '_' cim-class-name](http://schemas.dmtf.org/wbem/wscim/ wscim-major-version /cim-schema/ cim-schema-major-

 1145 version /' cim-class-schema '_' cim-class-name)

1146 Where:

- 1147 – [wscim-major-version](#) is the major version number of this specification. Note that this
- 1148 version number changes only if there are incompatible changes in the specification.
- 1149 – [cim-schema-major-version](#) is the major version number of the CIM schema version to
- 1150 which the class being converted belongs. Note that this version number changes only if
- 1151 there are incompatible changes in the CIM schema.
- 1152 • The location of the specific schema used to construct the instance should be declared by use of
- 1153 the `xsi:schemaLocation` attribute where the namespace URI and the specific class schema
- 1154 document URI location are combined as the value of the attribute, separated by whitespace.
- 1155 This provides a means for a recipient of the instance to determine which version of CIM defines
- 1156 the structure of this instance.

1157 For example:

1158 If the instance document is constructed under CIM schema version 2.11.0 and WS-CIM

1159 mapping specification 1.3.0, the following `xsi:schemaLocation` attribute should be specified in

1160 the instance document (where *ClassName* is the name of the CIM class of the instance):

```
1161 xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/ClassName
```

```
1162 http://schemas.dmtf.org/wbem/wscim/1.3.0/cim-schema/2.11.0/ClassName.xsd"
```

1163 If the URI in the attribute value can be de-referenced, then strict XML schema validation can be achieved.

1164 9.7 Superclass Class Representation

1165 The CIM Schema defines CIM classes in subclass / superclass relationships (hierarchies). For example,

1166 MOF reflects this structure in the definition of the class. A MOF statement of the following form defines

1167 `CIM_ClassA` as a subclass of the superclass `CIM_ClassB`:

```
1168 class CIM_ClassA : CIM_ClassB
```

1169 A MOF statement of the following form defines `CIM_ClassC` as a top-level class with no superclass:

```
1170 class CIM_ClassC
```

1171 Some management protocols may require a representation of the subclass/superclass hierarchy of a

1172 class as a value of an XML element in instance documents. The XSD mechanism by which to support

1173 representing this structure as a value of an XML element is provided in a separate schema that may be

1174 imported by protocols that require this capability for their instance documents.

1175 The immediate subclass/superclass relationship of a class shall be mapped to a single GED in the

1176 `classhierarchy` namespace according to the following rules:

- 1177 • Elements that describe the subclass / superclass relationships shall be placed in a separate
- 1178 schema from the WS-CIM class schema. For subclass / superclass relationships defined in the
- 1179 CIM Schema, the schema shall be named as follows:

```
1180 'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-
```

```
1181 major-version '/classhierarchy'
```

1182 Where:

- 1183 – [wscim-major-version](#) is the major version number of this specification. Note that this
- 1184 version number changes only if there are incompatible changes in the specification.
- 1185 – [cim-schema-major-version](#) is the major version number of the CIM schema version to
- 1186 which the class being converted belongs. Note that this version number changes only if
- 1187 there are incompatible changes in the CIM schema.

1188 This schema shall import the `http://schemas.dmtf.org/wbem/wscim/n/classhiertype` namespace, where 'n'
1189 represents the version number of this namespace.

1190 The class being mapped shall be represented by a GED whose name is derived from the name of the
1191 CIM class, appended with "_Class". This element shall be defined by an anonymous complex type that
1192 follows one of the following patterns:

- 1193 • The WS-CIM schema of a top-level class, `XXX_ClassC`, shall define the `_Class` element as a
1194 restriction of the `ctype:ClassHierarchyType`. The following pattern illustrates this rule:

```
1195 <xs:element name="XXX_ClassC_Class">
1196   <xs:complexType>
1197     <xs:complexContent>
1198       <xs:restriction base="ctype:ClassHierarchyType" />
1199     </xs:complexContent>
1200   </xs:complexType>
1201 </xs:element>
```

- 1202 • The WS-CIM schema of a subclass, `XXX_ClassA`, that is derived by a superclass,
1203 `XXX_ClassB`, shall restrict the `ctype:classHierarchyType` to contain a single element that
1204 references the corresponding `_Class` GED of the superclass. The following pattern illustrates
1205 this rule:

```
1206 <xs:element name="XXX_ClassA_Class">
1207   <xs:complexType>
1208     <xs:complexContent>
1209       <xs:restriction base="ctype:ClassHierarchyType">
1210         <xs:sequence>
1211           <xs:element ref="chier:XXX_ClassB_Class" />
1212         </xs:sequence>
1213       </xs:restriction>
1214     </xs:complexContent>
1215   </xs:complexType>
1216 </xs:element>
```

1217 See C.2.4 for an example classhierarchy schema.

1218 10 CIM Methods to WSDL Mappings

1219 This clause defines the structures that are necessary to define the messages and operation structures
1220 required for mapping a CIM method to WSDL.

1221 10.1 Defining WSDL Message Structures

1222 This clause provides the rules for creating WSDL message structures.

1223 The rules that govern the creation of WSDL message structures for a method are as follows:

- 1224 • The name of the WSDL input message should be the name of the CIM method plus
1225 `_InputMessage`. The following rules specify the structure of this element:
 - 1226 – The name of the `wsdl:part` element should be "body".
 - 1227 – The `element` attribute of the `wsdl:part` element shall specify the QName of the input
1228 message GED for the CIM method (see 9.5).

- 1229 • The name of the WSDL output message should be the name of the CIM method plus
1230 _OutputMessage. The following rules specify the structure of this element:
- 1231 – The name of the `wsdl:part` element should be "body".
- 1232 – The `element` attribute of the `wsdl:part` element shall specify the QName of the output
1233 message GED defined for the CIM method (see 9.5).

1234 EXAMPLE: The `wsdl:message` elements for the `RequestStateChange` CIM method (see 9.5) would
1235 be specified in the WSDL document as follows. The `wsdl:message` intended for input to the WSDL
1236 operation would be as follows (where the "class:" namespace prefix represents the namespace of the
1237 class whose interface is being exposed through this WSDL):

```
1238 <wsdl:message name="RequestStateChange_InputMessage">
1239   <wsdl:part name="body"
1240     element="class:RequestStateChange_INPUT"/>
1241 </wsdl:message>
```

1242 See C.3 for an example that shows the complete specification of the WSDL messages for this operation.

1243 10.2 Defining WSDL Operation Structures

1244 This specification defines *only* the structure of WSDL `portType` operations.

1245 The rules governing the structure of the WSDL operations used to invoke CIM methods are as follows:

- 1246 • The name attribute of the `wsdl:operation` element shall be the name of the CIM method.
- 1247 • The name attributes of the `wsdl:input` and `wsdl:output` child elements should be the name
1248 of the `wsdl:messages` that are referenced by these elements.
- 1249 • The message attributes of the `wsdl:input` and `wsdl:output` elements shall specify the
1250 QName of input and output message elements defined in 10.1.

1251 EXAMPLE: The `RequestStateChange` CIM method (see 9.5) is defined as follows:

```
1252 <wsdl:definitions
1253   ...
1254   xmlns:thisWSDL="http://. . .wsdl"
1255   ...>
1256 <wsdl:operation name="RequestStateChange">
1257   <wsdl:input name="RequestStateChange_InputMessage"
1258     message="thisWSDL:RequestStateChange_InputMessage"/>
1259   <wsdl:output name="RequestStateChange_OutputMessage"
1260     message="thisWSDL:RequestStateChange_OutputMessage"/>
1261 </wsdl:operation>
1262 </wsdl:definitions>
```

1263 This definition should be included in the `wsdl:portType` section of a WSDL document of a service that
1264 makes the CIM `RequestStateChange` method available to clients.

1265 10.3 Defining `wsa:Actions`

1266 The WS-Addressing specifications ([Web Services Addressing \(WS-Addressing\) 1.0 – Core](#) and [WS-Addressing](#))
1267 define the information model and syntax for the abstract messaging property Action. This
1268 property is defined as an IRI ([RFC 3987](#)), which identifies the semantics implied by a message (input,
1269 output, or fault). For the purposes of this specification, the Action property is restricted to a URI
1270 ([RFC 3986](#)) (a sequence of characters from a limited subset of the repertoire of US-ASCII characters).

1271 The details of how the action URI is specified on description artifacts are left to the specific protocol-
1272 binding specifications. Action URIs for faults are always left to the protocol-binding specifications.

1273 The rules for constructing WSA action URIs for input and output operation elements are as follows:

- 1274 • The action URI for an input message shall have the following form:

1275 class-namespace-name '/' input-name

1276 Where:

- 1277 – class-namespace-name is the full namespace name of the class being mapped as defined
1278 in 9.1.
- 1279 – input-name is the name of the CIM method.

- 1280 • The action URI for an output message shall have the following form:

1281 class-namespace-name '/' output-name 'Response'

1282 Where:

- 1283 – class-namespace-name is the full namespace name of the class being mapped as defined
1284 in 9.1.
- 1285 – output-name is the name of the output CIM method.

1286 EXAMPLE: The action URI for the input message of the `RequestStateChange` method is as follows:

```
1287 wsa:Action="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/EX_DerivedComponent/  
1288 RequestStateChange"
```

1289 The action URI for the output message of the `RequestStateChange` method is as follows:

```
1290 wsa:Action="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent/  
1291 RequestStateChangeResponse"
```

1292 See C.3 for an example that shows a complete mapping of the `RequestStateChange` method.

1293 11 Qualifier Mappings

1294 This clause defines the mapping of qualifiers to metadata fragments. The definition of the container for
1295 metadata fragments is left to the specific protocols to specify.

1296 11.1 General Format

1297 Rules for mapping qualifiers fall into two categories:

- 1298 • rules that describe the type definitions of qualifiers
- 1299 • rules that describe the XSD elements to which qualifiers are mapped

1300 This specification provides type definitions for the qualifier types used in CIM in `common.xsd` and
1301 mappings for all CIM qualifiers in the `qualifiers.xsd` (Annex A.2). It is expected that user-defined qualifiers
1302 follow the mapping rules for mapping qualifiers described in the following clauses.

1303 In addition to the mapping rules, user-defined qualifier mappings are governed by the following rules:

- 1304 • User-defined qualifiers shall not use any qualifier namespace defined in this specification.

1305 The qualifier types (types with names of the qualifier *Type*) defined in `common.xsd` should be used to
1306 define user-defined qualifiers. In the majority of cases, it is not necessary to create a new type definition
1307 to define a qualifier.

1308 **11.1.1 Single-Valued Qualifiers**

1309 This clause describes the rules for mapping single-valued qualifiers.

1310 Single-valued qualifiers that have an effective value that matches their default value shall not be mapped
1311 to a metadata fragment.

1312 The rules for mapping single-valued qualifiers that have an effective value that is not their global default
1313 value are as follows:

- 1314 • The rules for defining the type of a single-valued qualifier are as follows:
 - 1315 – Single-valued qualifiers shall be mapped to a complex type that is an extension of a simple
1316 content.
 - 1317 – The base type of this complex type shall correspond to the type of qualifier according to the
1318 datatype conversion rules defined in clause 8.
 - 1319 – The complex type shall extend the base type with a Boolean attribute of the name
1320 `qualifier`. This attribute is defined in the `cim` namespace. This attribute shall be
1321 specified with the XML Schema attribute `use="required"`.
- 1322 • The rules for mapping a single-valued qualifier are as follows:
 - 1323 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of
1324 the qualifier and which has been defined by the preceding rule. The name of the element
1325 shall be the name of the qualifier.
 - 1326 – The value of the `cim:qualifier` attribute in the mapped metadata fragment shall be `true`
1327 in all mappings.
 - 1328 – Single-valued qualifiers implicitly have the multiplicity `minOccurs="0" maxOccurs="1"`.
1329 Protocols should provide a mechanism by which to express this multiplicity in the schema
1330 of the document that contains generated metadata fragments.

- 1331 • The format of a schema element for defining a single-valued qualifier is as follows:

```
1332 '<xs:element name="" cim:qualifier-name "" type="" qualifier-type ""/>'
```

1333 Where:

- 1334 – `qualifier type` is typically one of the `qualifierType` types defined in `common.xsd` (but may be
1335 a user-defined type that complies with the mapping rules).

1336 `cim:qualifier-name` is the name of the qualifier, qualified by its namespace prefix.

1337 EXAMPLE: A generalized example of the mapping of a single-valued qualifier is as follows:

```
1338 <ns:QualifierName cim:qualifier="true">
1339   value
1340 </ns:QualifierName>
```

1341 Where:

- 1342 – `QualifierName` is the name of the qualifier.
- 1343 – `ns` is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1344 – `value` is the string representation of the qualifier value.

1345 For example, the mapping of the CIM qualifier `Abstract` is as follows:

```
1346 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
```

1347 **11.1.2 Multi-Valued Qualifiers**

1348 This clause describes the rules for mapping multi-valued qualifiers.

1349 Multi-valued qualifiers are qualifiers that are arrays. Multi-valued qualifiers that have an effective value
1350 that matches their default value shall not be mapped to a metadata fragment.

1351 The rules for mapping multi-valued qualifiers that have an effective value that is not their global default
1352 value are as follows:

- 1353 • The rules for defining the type of a multi-valued qualifier are as follows:
 - 1354 – Multi-valued qualifiers shall be mapped to a complex type that is an extension of a complex
1355 content.
 - 1356 – The base type of this complex type shall correspond to the single-valued qualifier *Type* of
1357 the member elements from which the qualifier array is constructed.
- 1358 • The rules for mapping a multi-valued qualifier are as follows:
 - 1359 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of
1360 the qualifier and which has been defined by the preceding rule. The *name* of the element
1361 shall be the same as that of the qualifier itself.
 - 1362 – The value of the `cim:qualifier` attribute in the metadata fragments shall be `true` in all
1363 mappings.
 - 1364 – Multi-valued qualifiers implicitly have the default multiplicity `minOccurs="0"`
1365 `maxOccurs="unbounded"`. Qualifier declarations may explicitly set different bounds on
1366 an array qualifier. Protocols should provide a mechanism by which to express the size of
1367 an array qualifier in the schema of the document that contains generated metadata
1368 fragments. The `minOccurs` and `maxOccurs` values shall correspond either to the default
1369 values if no qualifier array size is declared or to the declared qualifier array size.

1370 NOTE: The `common.xsd` file defines a string array of qualifier values, `qualifierSArray`. This definition complies
1371 with the first mapping rule in this clause. Therefore, in the majority of cases, it is not necessary to create a new array
1372 complex type definition to define a multi-valued qualifier. Rather, it is sufficient to use the `qualifierSArray` type
1373 defined in `common.xsd`.

1374 The format for a schema for defining a multi-valued qualifier is as follows:

```
1375 <xs:element name="" cim:qualifier-name "" type="" qualifier-array-type ""/>
```

1376 Where:

- 1377 – `cim:qualifier-name` is the name of the qualifier, qualified by its namespace prefix.
- 1378 – `qualifier-array-type` is typically the `qualifierSArray` type defined in the `common.xsd` file
1379 (but may be a user-defined type that complies with the mapping rules).

1380 EXAMPLE: A generalized example of the mapping of a multi-valued qualifier is as follows:

```
1381 <ns:QualifierName cim:qualifier="true">
1382   value
1383 </ns:QualifierName>
1384 ... // repeat QualifierName elements for each member of the array
```

1385 Where:

- 1386 – `QualifierName` is the name of the qualifier.
- 1387 – `ns` is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1388 – `n` is a sequential integer that represents the position of the entry in the array.
- 1389 – `value` is the string representation of the qualifier value.

1390 For example, the mapping of a ValueMap qualifier containing three entries ("OK", "Error", "Unknown") is
 1391 as follows:

```
1392 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
1393 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
1394 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
```

1395 A complete mapping of all qualifiers is provided in qualifiers.xsd (Annex A.2).

1396 **11.2 Mapping CIM Qualifiers to XSD Elements**

1397 All qualifiers are mapped using the normative rules in 11.1. The qualifiers listed in Table 8 are also
 1398 mapped directly into XSD features.

1399 **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).
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: <pre data-bbox="714 1178 1419 1474"><xs:element name="PropName"> <xs:complexType> <xs:simpleContent> <xs:restriction base="cim:cimString"> <xs:maxLength value="1024"/> </xs:restriction> </xs:simpleContent> </xs:complexType> </xs:element></pre>

CIM Qualifier	MOF Example	Mapped to XSD Structure
MaxValue	[MaxValue (100)]	<p>Mapped to a restriction using <code>xs:maxInclusive</code> on an integer datatype. Required, with the following exception:</p> <p>A qualifier value of NULL, which indicates the largest value allowed by the type, should not be mapped.</p> <p>For example:</p> <pre data-bbox="716 436 1421 720"> <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> </pre>
MinLen	[MinLen (10)]	<p>Mapped to a restriction using <code>xs:minLength</code> on a string datatype. Required, with the following exception:</p> <p>A qualifier value of 0 should not be mapped.</p> <p>For example:</p> <pre data-bbox="716 898 1421 1182"> <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> </pre>
MinValue	[MinValue (10)]	<p>Mapped to a restriction using <code>xs:minInclusive</code> on an integer datatype. Required, with the following exception:</p> <p>A qualifier value of NULL, which indicates the smallest value allowed by the type, should not be mapped.</p> <p>For example:</p> <pre data-bbox="716 1381 1421 1665"> <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> </pre>
OctetString uint8[] string[]	[OctetString]	<p>Normatively defined in 9.2.4.1 <code>cim:cimBase64Binary</code> <code>cim:cimHexBinary</code> array</p>
OUT	[OUT]	<p>The CIM output parameter is mapped to an element in the complex type for the output message GED (normatively defined in 9.5.1).</p>

CIM Qualifier	MOF Example	Mapped to XSD Structure
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): <ul style="list-style-type: none"> For non-overridden properties, <code>Override(NULL)</code>, 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]	<code>nillable="false"</code> (normatively defined in 9.2.1.1) NOTE: If the effective value of the <code>Required</code> qualifier is false, then <code>nillable="true"</code> (required).
ValueMap	[ValueMap (...)]	ValueMaps may be mapped to XSD as an enumeration (see 9.2.3).

1400 **11.3 Inheritance of Qualifiers**

1401 In addition to inheritance of properties, references, and methods through class inheritance, qualifier
 1402 values on any CIM elements are inherited. However, qualifiers are subject to special rules of inheritance.
 1403 Qualifier inheritance behavior is defined by the Flavors associated with a particular qualifier declaration.

1404 The rules covering qualifier inheritance are summarized in the third column of Table 9. In Table 9, the
 1405 term "overriding CIM elements in any subclasses" refers to CIM properties, references, and methods that
 1406 override other occurrences of the properties, references, or methods in their superclasses and therefore
 1407 form an inheritance chain. Note that duplicate property, reference, or method names that are *not*
 1408 overridden interrupt the inheritance chain for these CIM elements to their superclasses.

1409 **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: <code>Abstract</code>	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: <code>MaxLen</code>	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

FLAVOR	Qualifier Inheritance Behavior (Informative)	Metadata Fragment Mapping Behavior
		superclass.
ToSubclass: DisableOverride	<p>The qualifier value is inherited by any subclasses or overriding CIM elements in these subclasses.</p> <p>The value of the qualifier must not be changed in a subclass.</p> <p>EXAMPLE: <code>Key</code></p>	<p>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.</p>

ANNEX A (Informative)

Schemas

1410
1411
1412
1413

1414 This annex provides examples of the WS-CIM Schema (DSP8004), the Qualifiers Schema (DSP8005),
1415 and the Class Hierarchy Type Schema (DSP8006).

1416 A.1 Common WS-CIM Schema: DSP8004

1417 This schema contains common definitions.

```

1418 <?xml version="1.0" encoding="utf-8" ?>
1419 <xs:schema targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1420   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1421   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1422   elementFormDefault="qualified">
1423
1424 <!-- The following are runtime attribute definitions -->
1425   <xs:attribute name="Key" type="xs:boolean"/>
1426
1427   <xs:attribute name="Version" type="xs:string"/>
1428
1429 <!-- The following section defines the extended WS-CIM datatypes -->
1430
1431   <xs:complexType name="cimDateTime">
1432     <xs:choice>
1433       <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
1434       <xs:element name="Interval" type="xs:duration"/>
1435       <xs:element name="Date" type="xs:date" />
1436       <xs:element name="Time" type="xs:time" />
1437       <xs:element name="Datetime" type="xs:dateTime"/>
1438     </xs:choice>
1439     <xs:anyAttribute namespace="##any" processContents="lax"/>
1440   </xs:complexType>
1441
1442   <xs:complexType name="cimUnsignedByte">
1443     <xs:simpleContent>
1444       <xs:extension base="xs:unsignedByte">
1445         <xs:anyAttribute namespace="##any" processContents="lax"/>
1446       </xs:extension>
1447     </xs:simpleContent>
1448   </xs:complexType>
1449
1450   <xs:complexType name="cimByte">
1451     <xs:simpleContent>
1452       <xs:extension base="xs:byte">
1453         <xs:anyAttribute namespace="##any" processContents="lax"/>
1454       </xs:extension>
1455     </xs:simpleContent>
1456   </xs:complexType>
1457
1458   <xs:complexType name="cimUnsignedShort">
1459     <xs:simpleContent>
1460       <xs:extension base="xs:unsignedShort">

```

```
1461     <xs:anyAttribute namespace="##any" processContents="lax"/>
1462   </xs:extension>
1463 </xs:simpleContent>
1464 </xs:complexType>
1465
1466 <xs:complexType name="cimShort">
1467   <xs:simpleContent>
1468     <xs:extension base="xs:short">
1469       <xs:anyAttribute namespace="##any" processContents="lax"/>
1470     </xs:extension>
1471   </xs:simpleContent>
1472 </xs:complexType>
1473
1474 <xs:complexType name="cimUnsignedInt">
1475   <xs:simpleContent>
1476     <xs:extension base="xs:unsignedInt">
1477       <xs:anyAttribute namespace="##any" processContents="lax"/>
1478     </xs:extension>
1479   </xs:simpleContent>
1480 </xs:complexType>
1481
1482 <xs:complexType name="cimInt">
1483   <xs:simpleContent>
1484     <xs:extension base="xs:int">
1485       <xs:anyAttribute namespace="##any" processContents="lax"/>
1486     </xs:extension>
1487   </xs:simpleContent>
1488 </xs:complexType>
1489
1490 <xs:complexType name="cimUnsignedLong">
1491   <xs:simpleContent>
1492     <xs:extension base="xs:unsignedLong">
1493       <xs:anyAttribute namespace="##any" processContents="lax"/>
1494     </xs:extension>
1495   </xs:simpleContent>
1496 </xs:complexType>
1497
1498 <xs:complexType name="cimLong">
1499   <xs:simpleContent>
1500     <xs:extension base="xs:long">
1501       <xs:anyAttribute namespace="##any" processContents="lax"/>
1502     </xs:extension>
1503   </xs:simpleContent>
1504 </xs:complexType>
1505
1506 <xs:complexType name="cimString">
1507   <xs:simpleContent>
1508     <xs:extension base="xs:string">
1509       <xs:anyAttribute namespace="##any" processContents="lax"/>
1510     </xs:extension>
1511   </xs:simpleContent>
1512 </xs:complexType>
1513
1514 <xs:complexType name="cimBoolean">
1515   <xs:simpleContent>
1516     <xs:extension base="xs:boolean">
1517       <xs:anyAttribute namespace="##any" processContents="lax"/>
1518     </xs:extension>
```

```

1519     </xs:simpleContent>
1520 </xs:complexType>
1521
1522 <xs:complexType name="cimFloat">
1523   <xs:simpleContent>
1524     <xs:extension base="xs:float">
1525       <xs:anyAttribute namespace="##any" processContents="lax"/>
1526     </xs:extension>
1527   </xs:simpleContent>
1528 </xs:complexType>
1529
1530 <xs:complexType name="cimDouble">
1531   <xs:simpleContent>
1532     <xs:extension base="xs:double">
1533       <xs:anyAttribute namespace="##any" processContents="lax"/>
1534     </xs:extension>
1535   </xs:simpleContent>
1536 </xs:complexType>
1537
1538 <xs:complexType name="cimChar16">
1539   <xs:simpleContent>
1540     <xs:restriction base="cim:cimString">
1541       <xs:maxLength value="1"/>
1542       <xs:anyAttribute namespace="##any" processContents="lax"/>
1543     </xs:restriction>
1544   </xs:simpleContent>
1545 </xs:complexType>
1546
1547 <xs:complexType name="cimBase64Binary">
1548   <xs:simpleContent>
1549     <xs:extension base="xs:base64Binary">
1550       <xs:anyAttribute namespace="##any" processContents="lax"/>
1551     </xs:extension>
1552   </xs:simpleContent>
1553 </xs:complexType>
1554
1555 <xs:complexType name="cimHexBinary">
1556   <xs:simpleContent>
1557     <xs:extension base="xs:hexBinary">
1558       <xs:anyAttribute namespace="##any" processContents="lax"/>
1559     </xs:extension>
1560   </xs:simpleContent>
1561 </xs:complexType>
1562
1563 <xs:complexType name="cimAnySimpleType">
1564   <xs:simpleContent>
1565     <xs:extension base="xs:anySimpleType">
1566       <xs:anyAttribute namespace="##any" processContents="lax"/>
1567     </xs:extension>
1568   </xs:simpleContent>
1569 </xs:complexType>
1570
1571 <xs:complexType name="cimReference">
1572   <xs:sequence>
1573     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
1574   </xs:sequence>
1575   <xs:anyAttribute namespace="##any" processContents="lax"/>
1576 </xs:complexType>

```

```

1577
1578 <!-- The following datatypes are used exclusively to define metadata fragments -->
1579 <xs:attribute name="qualifier" type="xs:boolean"/>
1580
1581 <xs:complexType name="qualifierString">
1582   <xs:simpleContent>
1583     <xs:extension base="cim:cimString">
1584       <xs:attribute ref="cim:qualifier" use="required"/>
1585     </xs:extension>
1586   </xs:simpleContent>
1587 </xs:complexType>
1588
1589 <xs:complexType name="qualifierBoolean">
1590   <xs:simpleContent>
1591     <xs:extension base="cim:cimBoolean">
1592       <xs:attribute ref="cim:qualifier" use="required"/>
1593     </xs:extension>
1594   </xs:simpleContent>
1595 </xs:complexType>
1596
1597 <xs:complexType name="qualifierUInt32">
1598   <xs:simpleContent>
1599     <xs:extension base="cim:cimUnsignedInt">
1600       <xs:attribute ref="cim:qualifier" use="required"/>
1601     </xs:extension>
1602   </xs:simpleContent>
1603 </xs:complexType>
1604
1605 <xs:complexType name="qualifierSInt64">
1606   <xs:simpleContent>
1607     <xs:extension base="cim:cimLong">
1608       <xs:attribute ref="cim:qualifier" use="required"/>
1609     </xs:extension>
1610   </xs:simpleContent>
1611 </xs:complexType>
1612
1613 <xs:complexType name="qualifierSArray">
1614   <xs:complexContent>
1615     <xs:extension base="cim:qualifierString"/>
1616   </xs:complexContent>
1617 </xs:complexType>
1618
1619 <!-- The following element is to be used only for defining metadata -->
1620 <xs:element name="DefaultValue" type="xs:anySimpleType" />
1621
1622 </xs:schema>

```

1623 A.2 Qualifiers Schema: DSP8005

1624 The following schema is an example of the qualifiers schema that is based on CIM Schema 2.13.1.
 1625 Future versions of CIM Schema may add or delete qualifiers, which would be reflected in the
 1626 corresponding qualifiers.xsd file.

```

1627 <?xml version="1.0" encoding="utf-8" ?>
1628 <xs:schema
1629 targetNamespace="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1630   xmlns:cimQ="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1631   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"

```

```

1632     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1633     elementFormDefault="qualified">
1634
1635     <xs:import
1636         namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1637         schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1638
1639     <xs:element name="Abstract" type="cim:qualifierBoolean"/>
1640     <xs:element name="Aggregate" type="cim:qualifierBoolean"/>
1641     <xs:element name="Aggregation" type="cim:qualifierBoolean"/>
1642     <xs:element name="ArrayType" type="cim:qualifierString"/>
1643     <xs:element name="Association" type="cim:qualifierBoolean"/>
1644     <xs:element name="BitMap" type="cim:qualifierSArray"/>
1645     <xs:element name="BitValues" type="cim:qualifierSArray"/>
1646     <xs:element name="ClassConstraint" type="cim:qualifierSArray"/>
1647     <xs:element name="Counter" type="cim:qualifierBoolean"/>
1648     <xs:element name="Composition" type="cim:qualifierBoolean"/>
1649     <xs:element name="Deprecated" type="cim:qualifierSArray"/>
1650     <xs:element name="Description" type="cim:qualifierString"/>
1651     <xs:element name="DisplayName" type="cim:qualifierString"/>
1652     <xs:element name="DN" type="cim:qualifierBoolean"/>
1653     <xs:element name="EmbeddedInstance" type="cim:qualifierBoolean"/>
1654     <xs:element name="EmbeddedObject" type="cim:qualifierBoolean"/>
1655     <xs:element name="Exception" type="cim:qualifierBoolean"/>
1656     <xs:element name="Experimental" type="cim:qualifierBoolean"/>
1657     <xs:element name="Gauge" type="cim:qualifierBoolean"/>
1658     <xs:element name="In" type="cim:qualifierBoolean"/>
1659     <xs:element name="Indication" type="cim:qualifierBoolean"/>
1660     <xs:element name="Key" type="cim:qualifierBoolean"/>
1661     <xs:element name="MappingStrings" type="cim:qualifierSArray"/>
1662     <xs:element name="Max" type="cim:qualifierUInt32"/>
1663     <xs:element name="MethodConstraint" type="cim:qualifierSArray"/>
1664     <xs:element name="Min" type="cim:qualifierUInt32"/>
1665     <xs:element name="MaxLen" type="cim:qualifierUInt32"/>
1666     <xs:element name="MaxValue" type="cim:qualifierSInt64"/>
1667     <xs:element name="MinLen" type="cim:qualifierUInt32"/>
1668     <xs:element name="MinValue" type="cim:qualifierSInt64"/>
1669     <xs:element name="Revision" type="cim:qualifierString"/>           <!-- Is Deprecated -->
1670     <xs:element name="ModelCorrespondence" type="cim:qualifierSArray"/>
1671     <xs:element name="NullValue" type="cim:qualifierString"/>
1672     <xs:element name="OctetString" type="cim:qualifierBoolean"/>
1673     <xs:element name="Out" type="cim:qualifierBoolean"/>
1674     <xs:element name="Override" type="cim:qualifierString"/>
1675     <xs:element name="Propagated" type="cim:qualifierString"/>
1676     <xs:element name="PropertyConstraint" type="cim:qualifierSArray"/>
1677     <xs:element name="Read" type="cim:qualifierBoolean"/>
1678     <xs:element name="Required" type="cim:qualifierBoolean"/>
1679     <xs:element name="Schema" type="cim:qualifierString"/>
1680     <xs:element name="Static" type="cim:qualifierBoolean"/>
1681     <xs:element name="Terminal" type="cim:qualifierBoolean"/>
1682     <xs:element name="Units" type="cim:qualifierString"/>
1683     <xs:element name="UMLPackagePath" type="cim:qualifierString"/>
1684     <xs:element name="ValueMap" type="cim:qualifierSArray"/>

```

```

1685 <xs:element name="Values" type="cim:qualifierSArray"/>
1686 <xs:element name="Version" type="cim:qualifierString"/>
1687 <xs:element name="Weak" type="cim:qualifierBoolean"/>
1688 <xs:element name="Write" type="cim:qualifierBoolean"/>
1689
1690 <!-- Qualifier defined by DMTF for a future release of CIM Schema -->
1691 <!-- Included in this version at the request of the WSDM-CIM mapping team -->
1692 <xs:element name="Correlatable" type="cim:qualifierSArray"/>
1693
1694 <!-- Following qualifiers are considered to be "Optional Qualifiers" in CIM. -->
1695 <xs:element name="Alias" type="cim:qualifierString"/>
1696 <xs:element name="Delete" type="cim:qualifierBoolean"/>
1697 <xs:element name="Expensive" type="cim:qualifierBoolean"/>
1698 <xs:element name="IfDeleted" type="cim:qualifierBoolean"/>
1699 <xs:element name="Invisible" type="cim:qualifierBoolean"/>
1700 <xs:element name="Large" type="cim:qualifierBoolean"/>
1701 <xs:element name="Provider" type="cim:qualifierString"/>
1702 <xs:element name="PropertyUsage" type="cim:qualifierString"/>
1703 <xs:element name="Syntax" type="cim:qualifierString"/>
1704 <xs:element name="SyntaxType" type="cim:qualifierString"/>
1705 <xs:element name="TriggerType" type="cim:qualifierString"/>
1706 <xs:element name="UnknownValues" type="cim:qualifierSArray"/>
1707 <xs:element name="UnsupportedValues" type="cim:qualifierSArray"/>
1708
1709 </xs:schema>

```

1710 A.3 Class Hierarchy Type Schema: DSP8006

1711 The complex type definition in the following schema provides the type of GEDs that describe the CIM
 1712 Schema subclass / superclass hierarchy. The element `ClassHierarchy` may be used by protocols as
 1713 an element in XML instance documents of a CIM instance that contains a value representing the subclass
 1714 / superclass hierarchy of a class. Its presence as an element in an instance document would be covered
 1715 by the `xs:any` in the WS-CIM schema of the instance's class.

```

1716 <?xml version="1.0" encoding="utf-8"?>
1717 <xs:schema
1718   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1719   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1720   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1721   <xs:complexType name="ClassHierarchyType">
1722     <xs:sequence>
1723       <xs:any minOccurs="0" namespace="##any" processContents="lax" />
1724     </xs:sequence>
1725   </xs:complexType>
1726
1727   <xs:element name="ClassHierarchy" type="ctype:ClassHierarchyType"/>
1728
1729 </xs:schema>

```


ANNEX B (Informative)

Conventions

1730
1731
1732
1733

1734 In XML and MOF examples, an ellipsis (" . . . ") indicates omitted or optional entries that would typically
1735 occupy the position of the ellipsis.

1736 The following conventions are followed for defining formats of entries such as URIs:

- 1737 • Literal characters within a format definition are surrounded by single quotes.
- 1738 • Names of variables within a format are in standard text and are explicitly defined by means of a
1739 "Where: variable-name is ..." section that follows the format definition.
- 1740 • A specific value of a variable within a generalized example of a formatted entry is displayed in
1741 *italics*.
- 1742 • Definitions of formats are case sensitive.
- 1743 • Whitespace, if any, in formats is explicitly indicated.

1744 The following typographical conventions are used:

- 1745 • `Monospace font`: CIM datatypes and element names as well as XML and WSDL element
1746 and attribute names.
- 1747 • `Courier new 8, gray background`: Code examples

ANNEX C (Informative)

Examples

1748
1749
1750
1751

1752 This annex contains examples of converting MOF definitions of several classes into XML Schema, WSDL
1753 fragments, and metadata fragments. Although the classes are fictional creations used to illustrate
1754 different features of the conversion, the classes are based on actual CIM classes.

1755 C.1 MOF Definitions

1756 This clause contains the MOF definitions that are converted in these examples.

1757 C.1.1 EX_BaseComponent

```

1758     [Abstract, Version ( "2.x" ), Description (
1759         "EX_BaseComponent serves as an example base CIM class.")]
1760 class EX_BaseComponent {
1761     [Description (
1762         "A datetime value indicating when the object was installed.")]
1763     datetime InstallDate;
1764     [Description (
1765         "The Name property defines the label by which the object is "
1766         "known."),
1767         MaxLen ( 1024 ), Required]
1768     string Name;
1769     [Description (
1770         "A set of descriptive statements that can be used to describe the "
1771         "state of a Component."),
1772         ArrayType ( "Indexed" ) ]
1773     string StatusDescriptions[];
1774     [Description (
1775         "A descriptive code representing operational health of a Component."),
1776         ValueMap { "OK", "Error", "Unknown" }, MaxLen ( 10 ) ]
1777     string HealthStatus;
1778 };

```

1779 C.1.2 EX_DerivedComponent

```

1780     [Version ( "2.x" ), Description (
1781         "This class extends EX_BaseComponent.")]
1782 class EX_DerivedComponent : EX_BaseComponent {
1783     [Description (
1784         "EnabledState is an integer enumeration that indicates the "
1785         "enabled and disabled states of a derived Component."),
1786         ValueMap { "0", "1", "2", "3" },
1787         Values { "Unknown", "Other", "Enabled", "Disabled" } ]
1788     uint16 EnabledState=3;
1789     [Description (
1790         "Boolean flag indicating availability of a Component." ) ]
1791     boolean AvailableFlag;
1792     [Description (
1793         "Requests that the state of the element be changed to the "
1794         "value specified in the RequestedState parameter."),
1795         ValueMap { "0", "1", "2", "3..32767", "32768..65535" },
1796         Values { "Completed with No Error", "Not Supported",

```

```

1797         "Failed", "DMTF Reserved", "Vendor Specific" } ]
1798     uint32 RequestStateChange(
1799         [IN, Description (
1800             "The state requested for the Component."),
1801             ValueMap { "2", "3", "4" },
1802             Values { "Enabled", "Disabled" "Shutdown" } ]
1803     uint16 RequestedState,
1804     [IN ( false ), OUT, Description (
1805         "Reference to an instance of some class (undefined in this "
1806         "example) that is returned upon completion of the operation.")]
1807     CIM_SomeClass REF ResultClass,
1808     [IN, Description (
1809         "A timeout period that specifies the maximum amount of "
1810         "time that the client expects the transition to the new "
1811         "state to take. ")]
1812     datetime TimeoutPeriod);
1813 };

```

1814 C.1.3 EX_AssociationComponent

```

1815     [Association, Version ( "2.x" ), Description (
1816         "Indicates that two entites are associated.")]
1817 class EX_Association {
1818     [Key, Description (
1819         "AssociatingComponent represents one Component is "
1820         "associated with the Component referenced as AssociatedComponent.")]
1821     EX_BaseComponent REF AssociatingComponent;
1822     [Key, Description (
1823         "AssociatedComponent represents another Component (up to 4) that "
1824         "is associated with the Component referenced as "
1825         "AssociatingComponent."),
1826         Max ( 4 )]
1827     EX_BaseComponent REF AssociatedComponent;
1828     [Description (
1829         "The point in time that the Components were associated.")]
1830     datetime WhenAssociated;
1831     [Description (
1832         "Boolean indicating whether the association is maintained.")]
1833     boolean AssocMaintained;
1834 };

```

1835 C.2 XSD

1836 This clause shows the XML Schema files that would result from the application of this specification to the
 1837 preceding example CIM classes.

1838 C.2.1 EX_BaseComponent

```

1839 <?xml version="1.0" encoding="utf-8"?>
1840 <xs:schema
1841     targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1842     xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1843     xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1844     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1845     ...>
1846 <xs:import
1847     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1848     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1849 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>

```

```

1850 <xs:element name="Name">
1851   <xs:complexType>
1852     <xs:simpleContent>
1853       <xs:restriction base="cim:cimString">
1854         <xs:maxLength value="1024"/>
1855         <xs:anyAttribute namespace="##any" processContents="lax"/>
1856       </xs:restriction>
1857     </xs:simpleContent>
1858   </xs:complexType>
1859 </xs:element>
1860 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1861 <xs:element name="HealthStatus" nillable="true">
1862   <xs:complexType>
1863     <xs:simpleContent>
1864       <xs:restriction base="cim:cimString">.
1865         <xs:enumeration value="OK"/>
1866         <xs:enumeration value="Error"/>
1867         <xs:enumeration value="Unknown"/>
1868         <xs:maxLength value="10"/>
1869         <xs:anyAttribute namespace="##any" processContents="lax"/>
1870       </xs:restriction>
1871     </xs:simpleContent>
1872   </xs:complexType>
1873 </xs:element>
1874 <xs:complexType name="EX_BaseComponent_Type">
1875   <xs:sequence>
1876     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1877     <xs:element ref="class:InstallDate" minOccurs="0"/>
1878     <xs:element ref="class:Name"/>
1879     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1880     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1881   </xs:sequence>
1882   <xs:anyAttribute namespace="##any" processContent="lax"/>
1883 </xs:complexType>
1884 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type"/>
1885 </xs:schema>

```

1886 C.2.2 EX_DerivedComponent

```

1887 <?xml version="1.0" encoding="utf-8"?>
1888 <xs:schema
1889   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1890   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1891   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1892   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1893   ...>
1894 <xs:import
1895   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1896   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1897 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
1898 <xs:element name="Name">
1899   <xs:complexType>
1900     <xs:simpleContent>
1901       <xs:restriction base="cim:cimString">
1902         <xs:maxLength value="1024"/>
1903         <xs:anyAttribute namespace="##any" processContents="lax"/>
1904       </xs:restriction>
1905     </xs:simpleContent>
1906   <xs:complexType>

```

```

1907 </xs:element>
1908 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1909 <xs:element name="HealthStatus" nillable="true">
1910   <xs:complexType>
1911     <xs:simpleContent>
1912       <xs:restriction base="cim:cimString">
1913         <xs:enumeration value="OK"/>
1914         <xs:enumeration value="Error"/>
1915         <xs:enumeration value="Unknown"/>
1916         <xs:maxLength value="10"/>
1917         <xs:anyAttribute namespace="##any" processContents="lax"/>
1918       </xs:restriction>
1919     </xs:simpleContent>
1920   </xs:complexType>
1921 </xs:element>
1922 <xs:element name="EnabledState" nillable="true">
1923   <xs:complexType>
1924     <xs:simpleContent>
1925       <xs:restriction base="cim:cimUnsignedShort">
1926         <xs:enumeration value="0"/>
1927         <xs:enumeration value="1"/>
1928         <xs:enumeration value="2"/>
1929         <xs:enumeration value="3"/>
1930         <xs:anyAttribute namespace="##any" processContents="lax"/>
1931       </xs:restriction>
1932     </xs:simpleContent>
1933   </xs:complexType>
1934 </xs:element>
1935 <xs:element name="AvailableFlag" type="cim:cimBoolean" nillable="true"/>
1936 <xs:complexType name="EX_DerivedComponent_Type">
1937   <xs:sequence>
1938     <xs:element ref="class:AvailableFlag" minOccurs="0"/>
1939     <xs:element ref="class:EnabledState" minOccurs="0"/>
1940     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1941     <xs:element ref="class:InstallDate" minOccurs="0"/>
1942     <xs:element ref="class:Name"/>
1943     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1944     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1945   </xs:sequence>
1946   <xs:anyAttribute namespace="##any" processContent="lax"/>
1947 </xs:complexType>
1948 <xs:element name="EX_DerivedComponent" type="class:EX_DerivedComponent_Type"/>
1949 <xs:element name="RequestStateChange_INPUT">
1950   <xs:complexType>
1951     <xs:sequence>
1952       <xs:element name="RequestedState" nillable="true">
1953         <xs:complexType>
1954           <xs:simpleContent>
1955             <xs:restriction base="cim:cimUnsignedShort">
1956               <xs:enumeration value="2"/>
1957               <xs:enumeration value="3"/>
1958               <xs:enumeration value="4">
1959                 <xs:anyAttribute namespace="##any" processContents="lax"/>
1960               </xs:restriction>
1961             </xs:simpleContent>
1962           </xs:complexType>
1963         </xs:element>
1964       <xs:element name="TimeoutPeriod" type="cim:cimDateTime" nillable="true"/>

```

```

1965     </xs:sequence>
1966   </xs:complexType>
1967 </xs:element>
1968 <xs:element name="RequestStateChange_OUTPUT">
1969   <xs:complexType>
1970     <xs:sequence>
1971       <xs:element name="ResultClass" type="cim:cimReference" nillable="true"/>
1972       <xs:element name="ReturnValue" nillable="true">
1973         <xs:complexType>
1974           <xs:simpleContent>
1975             <xs:restriction base="cim:cimAnySimpleType">
1976               <xs:simpleType>
1977                 <xs:union>
1978                   <xs:simpleType>
1979                     <xs:restriction base="xs:unsignedInt">
1980                       <xs:enumeration value="0"/>
1981                       <xs:enumeration value="1"/>
1982                       <xs:enumeration value="2"/>
1983                     </xs:restriction>
1984                   </xs:simpleType>
1985                   <xs:simpleType>
1986                     <xs:restriction base="xs:unsignedInt">
1987                       <xs:minInclusive value="3"/>
1988                       <xs:maxInclusive value="32767"/>
1989                     </xs:restriction>
1990                   </xs:simpleType>
1991                   <xs:simpleType>
1992                     <xs:restriction base="xs:unsignedInt">
1993                       <xs:minInclusive value="32768"/>
1994                       <xs:maxInclusive value="65535"/>
1995                     </xs:restriction>
1996                   </xs:simpleType>
1997                 </xs:union>
1998               </xs:simpleType>
1999               <xs:anyAttribute namespace="##any" processContents="lax"/>
2000             </xs:restriction>
2001           </xs:simpleContent>
2002         </xs:complexType>
2003       </xs:element>
2004     </xs:sequence>
2005   </xs:complexType>
2006 </xs:element>
2007 </xs:schema>

```

2008 C.2.3 EX_AssociationComponent

```

2009 <?xml version="1.0" encoding="utf-8"?>
2010 <xs:schema
2011   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2012   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
2013   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2014   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2015   ...>
2016 <xs:import
2017   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
2018   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
2019 <xs:element name="AssociatingComponent" type="cim:cimReference"/>
2020 <xs:element name="AssociatedComponent" type="cim:cimReference"/>
2021 <xs:element name="WhenAssociated" type="cim:cimDateTime" nillable="true"/>

```

```

2022 <xs:element name="AssocMaintained" type="cim:cimBoolean" nillable="true"/>
2023 <xs:complexType name="EX_AssociationComponent_Type">
2024   <xs:sequence>
2025     <xs:element ref="class:AssociatedComponent" />
2026     <xs:element ref="class:AssociatingComponent" />
2027     <xs:element ref="class:AssocMaintained" minOccurs="0"/>
2028     <xs:element ref="class:WhenAssociated" minOccurs="0"/>
2029     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2030   </xs:sequence>
2031   <xs:anyAttribute namespace="##any" processContent="lax"/>
2032 </xs:complexType>
2033 <xs:element name="EX_AssociationComponent" type="class:EX_AssociationComponent_Type"/>
2034 </xs:schema>

```

2035 C.2.4 Class Hierarchy Schema

```

2036 <?xml version="1.0" encoding="utf-8"?>
2037 <xs:schema
2038   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2039   xmlns:chier="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2040   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2041   xmlns:xs="http://www.w3.org/2001/XMLSchema">
2042   <xs:import
2043     namespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2044     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/classhiertype.xsd"/>
2045   <xs:element name="EX_BaseComponent_Class">
2046     <xs:complexType>
2047       <xs:complexContent>
2048         <xs:restriction base="ctype:ClassHierarchyType" />
2049       </xs:complexContent>
2050     </xs:complexType>
2051   </xs:element>
2052   <xs:element name="EX_DerivedComponent_Class">
2053     <xs:complexType>
2054       <xs:complexContent>
2055         <xs:restriction base="ctype:ClassHierarchyType">
2056           <xs:sequence>
2057             <xs:element ref="chier:EX_BaseComponent_Class" />
2058           </xs:sequence>
2059         </xs:restriction>
2060       </xs:complexContent>
2061     </xs:complexType>
2062   </xs:element>
2063   <xs:element name="EX_AssociationComponent_Class">
2064     <xs:complexType>
2065       <xs:complexContent>
2066         <xs:restriction base="ctype:ClassHierarchyType" />
2067       </xs:complexContent>
2068     </xs:complexType>
2069   </xs:element>
2070 </xs:schema>

```

2071 C.3 WSDL Fragments

2072 This clause contains the WSDL fragments (`wsdl:message`, `wsdl:operation`) that would result from
 2073 the application of this specification to the `EX_DerivedComponent` class. This class specifies only one
 2074 method, `RequestStateChange`.

```

2075 <?xml version="1.0" encoding="utf-8"?>

```

```

2076 <wsdl:definitions
2077     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
2078     targetNamespace="http://. . .wsdl"
2079     xmlns:cimClass="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
2080     xmlns:thisWSDL="http://. . .wsdl"
2081     ...>
2082 <w:import namespace="http://www.w3.org/2005/08/addressing"
2083     location="http://www.w3.org/2005/08/addressing/ws-addr.xsd"/>
2084 <wsdl:types>
2085     ... <!-- Schema of EX_DerivedComponent -->
2086 </wsdl:types>
2087 <wsdl:message name="RequestStateChange_InputMessage">
2088     <wsdl:part name="body"
2089         element="cimClass:RequestStateChange_INPUT"/>
2090 </wsdl:message>
2091 <wsdl:message name="RequestStateChange_OutputMessage">
2092     <wsdl:part name="body"
2093         element="cimClass:RequestStateChange_OUTPUT"/>
2094 </wsdl:message>
2095 <!-- OPERATION: RequestStateChange
2096     <wsdl:operation name="RequestStateChange">
2097         <wsdl:input name="RequestStateChange_InputMessage"
2098             message="thisWSDL:RequestStateChange_InputMessage"
2099         <wsdl:output name="RequestStateChange_OutputMessage"
2100             message="thisWSDL:RequestStateChange_OutputMessage"
2101         </wsdl:operation>
2102 -->
2103 </wsdl:definitions>

```

2104 C.4 MetaData Fragments

2105 Metadata fragments are generated from the qualifiers that are associated with a class, property,
2106 reference, method, or parameter. XML documents that incorporate these fragments must import the cim
2107 and cimQ namespaces.

2108 C.4.1 EX_BaseComponent

2109 C.4.1.1 Class Qualifiers

```

2110 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
2111 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2112 <cimQ:Description cim:qualifier="true">
2113     EX_BaseComponent serves as an example base CIM class.
2114 </cimQ:Description>

```

2115 C.4.1.2 Property Qualifiers

2116 C.4.1.2.1 HealthStatus

```

2117 <cimQ:Description cim:qualifier="true">
2118     A descriptive code of the operational health of a Component.
2119 </cimQ:Description>
2120 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
2121 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>

```

2122 C.4.1.2.2 InstallDate

```

2123 <cimQ:Description cim:qualifier="true">
2124     EX_BaseComponent serves as an example base CIM class.
2125 </cimQ:Description>
2126 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>

```


2127 C.4.1.2.3 Name

```

2128 <cimQ:Description cim:qualifier="true">
2129     The Name property defines the label by which the object is known.
2130 </cimQ:Description>
2131 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
2132 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

```

2133 C.4.1.2.4 StatusDescriptions

```

2134 <cimQ:Description cim:qualifier="true">
2135     A set of descriptive statements that can be used to describe the state of an Component.
2136 </cimQ:Description>
2137 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

```

2138 C.4.2 EX_DerivedComponent**2139 C.4.2.1 Class Qualifiers**

```

2140 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2141 <cimQ:Description cim:qualifier="true">
2142     This class extends EX_BaseComponent.
2143 </cimQ:Description>

```

2144 C.4.2.2 Property Qualifiers**2145 C.4.2.2.1 AvailableFlag**

```

2146 <cimQ:Description cim:qualifier="true">
2147     Boolean flag indicating availability of a Component.

```

2148 C.4.2.2.2 EnabledState

```

2149 <cimQ:Description cim:qualifier="true">
2150     EnabledState is an integer enumeration that indicates the enabled
2151     and disabled states of a derived Component.
2152 </cimQ:Description>
2153 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
2154 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
2155 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
2156 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
2157 <cimQ:Values cim:qualifier="true">Unknown</cimQ:Values>
2158 <cimQ:Values cim:qualifier="true">Other</cimQ:Values>
2159 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
2160 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
2161 <cim:DefaultValue xsi:type="xs:uint16">3</cim:DefaultValue>
2162 HealthStatus
2163 <cimQ:Description cim:qualifier="true">
2164     A descriptive code of the operational health of a Component.
2165 </cimQ:Description>
2166 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
2167 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
2168 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
2169 </cimQ:Description>

```

2170 C.4.2.2.3 InstallDate

```

2171 <cimQ:Description cim:qualifier="true">
2172     EX_BaseComponent serves as an example base CIM class.
2173 </cimQ:Description>

```

2174 C.4.2.2.4 Name

```

2175 <cimQ:Description cim:qualifier="true">
2176     The Name property defines the label by which the object is known.
2177 </cimQ:Description>
2178 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
2179 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

```

2180 C.4.2.2.5 StatusDescriptions

```

2181 <cimQ:Description cim:qualifier="true">
2182     A set of descriptive statements that can used to describe the state of an Component.
2183 </cimQ:Description>
2184 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

```

2185 C.4.2.2.6 AvailableFlag

```

2186 <cimQ:Description cim:qualifier="true">
2187     Boolean flag indicating availability of a Component.
2188 </cimQ:Description>

```

2189 C.4.2.3 Method and Parameter Qualifiers**2190 C.4.2.3.1 RequestStatusChange Method**

```

2191 <cimQ:Description cim:qualifier="true">
2192     Requests that the state of the element be changed to the value
2193     specified in the RequestedState parameter.
2194 </cimQ:Description>
2195 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
2196 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
2197 <cimQ:ValueMap cim:qualifier="true">..</cimQ:ValueMap>
2198 <cimQ:ValueMap cim:qualifier="true">4096</cimQ:ValueMap>
2199 <cimQ:ValueMap cim:qualifier="true">4100..32767</cimQ:ValueMap>
2200 <cimQ:ValueMap cim:qualifier="true">32768..65535</cimQ:ValueMap>
2201 <cimQ:Values cim:qualifier="true">Completed with No Error</cimQ:Values>
2202 <cimQ:Values cim:qualifier="true">Not Supported</cimQ:Values>
2203 <cimQ:Values cim:qualifier="true">Unknown or Unspecified Error</cimQ:Values>
2204 <cimQ:Values cim:qualifier="true">Failed</cimQ:Values>
2205 <cimQ:Values cim:qualifier="true">DMTF Reserved</cimQ:Values>
2206 <cimQ:Values cim:qualifier="true">Vendor Specific</cimQ:Values>

```

2207 C.4.2.3.2 RequestedState Parameter

```

2208 <cimQ:Description cim:qualifier="true">
2209     The state requested for the Component.
2210 </cimQ:Description>
2211 <cimQ:In cim:qualifier="true">true</cimQ:In>
2212 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
2213 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
2214 <cimQ:ValueMap cim:qualifier="true">4</cimQ:ValueMap>
2215 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
2216 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
2217 <cimQ:Values cim:qualifier="true">Shutdown</cimQ:Values>

```

2218 C.4.2.3.3 ResultClass Parameter

```

2219 <cimQ:Description cim:qualifier="true">
2220     Reference to an instance of some class (undefined in this example)
2221     that is returned upon completion of the operation.

```

```

2222 </cimQ:Description>
2223 <cimQ:Out cim:qualifier="true">true</cimQ:Out>
2224 <cimQ:In cim:qualifier="true">false</cimQ:In>

```

2225 C.4.2.3.4 TimeoutPeriod Parameter

```

2226 <cimQ:Description cim:qualifier="true">
2227     A timeout period that specifies the maximum amount of time that the
2228     client expects the transition to the new state to take.
2229 </cimQ:Description>
2230 <cimQ:In cim:qualifier="true">true</cimQ:In>

```

2231 C.4.3 EX_AssociationComponent

2232 C.4.3.1 Class Qualifiers

```

2233 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2234 <cimQ:Description cim:qualifier="true">
2235     Indicates that two entites are associated.
2236 </cimQ:Description>
2237 <cimQ:Association cim:qualifier="true">true</cimQ:Association>

```

2238 C.4.3.2 Property Qualifiers

2239 C.4.3.2.1 AssociatedComponent

```

2240 <cimQ:Key cim:qualifier="true">true</cimQ:Key>
2241 <cimQ:Description cim:qualifier="true">
2242     AssociatedComponent represents another Component (up to 4) that is associated
2243     with the Component referenced as AssociatingComponent.
2244 </cimQ:Description>
2245 <cimQ:Max cim:qualifier="true">4</cimQ:Max>

```

2246 C.4.3.2.2 AssociatingComponent

```

2247 <cimQ:Key cim:qualifier="true">true</cimQ:Key>
2248 <cimQ:Description cim:qualifier="true">
2249     An AssociatingComponent represents one Component is associated with the
2250     component referenced as AssociatedComponent.
2251 </cimQ:Description>

```

2252 C.4.3.2.3 AssocMaintained

```

2253 <cimQ:Description cim:qualifier="true">
2254     Boolean indicating whether the association is maintained.
2255 </cimQ:Description>

```

2256 C.4.3.2.4 WhenAssociated

```

2257 <cimQ:Description cim:qualifier="true">
2258     The point in time that the Components were associated.
2259 </cimQ:Description>

```

2260