



1
2
3
4

Document Number: DSP0230

Date: 2010-02-08

Version: 1.0.2

5 **WS-CIM Mapping Specification**

6 **Document Type: Specification**
7 **Document Status: DMTF Standard**
8 **Document Language: E**
9

10 Copyright notice

11 Copyright © 2007, 2010 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, provided that correct attribution is given. As DMTF specifications may be revised from time to
15 time, the particular version and release date should always be noted.

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26 withdrawn or modified after publication, and shall be indemnified and held harmless by any party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

29 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
30 such patent may relate to or impact implementations of DMTF standards, visit
31 <http://www.dmtf.org/about/policies/disclosures.php>.

32

CONTENTS

34	Foreword	5
35	Introduction	6
36	1 Scope	7
37	1.1 In-Scope Features	7
38	1.2 Out-of-Scope Considerations	7
39	2 Conformance	8
40	3 Normative References	8
41	4 Terms and Definitions	9
42	5 Symbols and Abbreviated Terms	10
43	6 Namespace Prefixes and Schema Locations	11
44	7 Dereferencing Schema URI Locations in Order to Access XML Schema	13
45	8 Mapping Primitive Datatypes	14
46	8.1 cimDateTime Datatype	15
47	8.2 CIM References	16
48	8.3 cimAnySimpleType Datatype	17
49	8.4 Derivation by Restriction	17
50	8.5 WS-CIM Canonical Values	17
51	9 CIM Class to XML Schema Mappings	18
52	9.1 Class Namespace	18
53	9.2 Properties	18
54	9.3 Class Structure	27
55	9.4 Class Inheritance	29
56	9.5 Method Parameters	30
57	9.6 CIM Instances	33
58	9.7 Superclass Class Representation	34
59	10 CIM Methods to WSDL Mappings	35
60	10.1 Defining WSDL Message Structures	35
61	10.2 Defining WSDL Operation Structures	36
62	10.3 Defining wsa:Actions	36
63	11 Qualifier Mappings	37
64	11.1 General Format	37
65	11.2 Mapping CIM Qualifiers to XSD Elements	39
66	11.3 Inheritance of Qualifiers	42
67	ANNEX A (Informative) Schemas	43
68	A.1 Common WS-CIM Schema: DSP8004	43
69	A.2 Qualifiers Schema: DSP8005	46
70	A.3 Class Hierarchy Type Schema: DSP8006	48
71	ANNEX B (Informative) Examples	49
72	B.1 MOF Definitions	49
73	B.2 XSD	50
74	B.3 WSDL Fragments	55
75	B.4 MetaData Fragments	55
76	ANNEX C (Informative) Collation Optimization Available to Implementers	59
77	C.1 Sorting with Limited Character Set	59
78	C.2 Note of Caution Concerning Collation	59
79	ANNEX D (Informative) Change Log	60
80	Bibliography	61
81		

82 Tables

83	Table 1 – Namespaces	11
84	Table 2 – Namespace Prefixes	12
85	Table 3 – Schema URI Locations	12
86	Table 4 – XSD DSP Numbers.....	12
87	Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes	14
88	Table 6 – Rules for Converting datetime to cimDateTime	15
89	Table 7 – Rules for Converting cimDateTime to datetime	15
90	Table 8 – CIM Qualifiers Mapped to XSD Elements.....	40
91	Table 9 – Rules of Qualifier Inheritance.....	42
92		

93

Foreword

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

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

98 Authors:

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

118

119

Introduction

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

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

132 Document Conventions

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

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

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

143 The following typographical conventions are used:

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

147

148

WS-CIM Mapping Specification

149 1 Scope

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

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

159 1.1 In-Scope Features

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

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

167 1.2 Out-of-Scope Considerations

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

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

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

192 **2 Conformance**

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

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

197 **3 Normative References**

198 The following referenced documents are indispensable for the application of this document. For dated or
199 versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies.
200 For references without a date or version, the latest published edition of the referenced document
201 (including any corrigenda or DMTF update versions) applies.

202 DMTF DSP0004, *Common Information Model (CIM) Infrastructure Specification 2.5*,
203 http://www.dmtf.org/standards/published_documents/DSP0004_2.5.pdf

204 DMTF DSP0226, *Web Services for Management Specification 1.1*,
205 http://www.dmtf.org/standards/published_documents/DSP0226_1.1.pdf

206 DMTF DSP8004, *WS-CIM Common XSD 1.0*,
207 <http://schemas.dmtf.org/wbem/wscim/1/common.xsd>

208 DMTF DSP8005, *WS-CIM Qualifiers XSD 1.0*,
209 <http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/qualifiers.xsd>

210 DMTF DSP8006, *WS-CIM ClassHierType XSD 1.0*,
211 <http://schemas.dmtf.org/wbem/wscim/1/classhiertype.xsd>

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

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

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

218 ISO, ISO/IEC 10646:2003 *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*,
219 [http://standards.iso.org/ittf/PubliclyAvailableStandards/c039921_ISO_IEC_10646_2003\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/c039921_ISO_IEC_10646_2003(E).zip)

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

222 W3C, *Namespaces in XML 1.0 (Third Edition)*, W3C Recommendation, 8 December 2009,
223 <http://www.w3.org/TR/REC-xml-names>

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

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

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

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

232 **4 Terms and Definitions**

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

234 **4.1**

235 **can**

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

237 **4.2**

238 **cannot**

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

240 **4.3**

241 **conditional**

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

244 **4.4**

245 **mandatory**

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

248 **4.5**

249 **may**

250 indicates a course of action permissible within the limits of the document

251 **4.6**

252 **need not**

253 indicates a course of action permissible within the limits of the document

254 **4.7**

255 **optional**

256 indicates a course of action permissible within the limits of the document

257 **4.8**

258 **shall**

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

261 **4.9**

262 **shall not**

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

- 265 **4.10**
266 **should**
267 indicates that among several possibilities, one is recommended as particularly suitable, without
268 mentioning or excluding others, or that a certain course of action is preferred but not necessarily required
- 269 **4.11**
270 **should not**
271 indicates that a certain possibility or course of action is deprecated but not prohibited
- 272 **4.12**
273 **Global Element Declaration**
274 element declaration in an XML Schema that places the element as an immediate child of the root element
275 of the schema
- 276 **4.13**
277 **Managed Object Format**
278 an IDL based language, defined by the DMTF, expressing the structure, behavior, and semantics of a
279 CIM class and its instances
- 280 **4.14**
281 **Uniform Record Identifier**
282 a Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or
283 physical resource. See [rfc3986](#).
- 284 **4.15**
285 **Runtime**
286 describes the situation where XML instances are produced that are conformant to this specification
- 287 **4.16**
288 **WS-CIM**
289 of or pertaining to this specification
290 NOTE: "WS-CIM" is not an acronym; it should be treated simply as the name of the contents of the specification.
- 291 **4.17**
292 **XML instance document**
293 an XML document that conforms to a specified XML Schema
294 As used in this specification, *XML instance document* refers to a document that conforms to an XML
295 Schema that conforms to the rules in clause 9.
- 296 **4.18**
297 **XSDL**
298 offers facilities for describing the structure and constraining the contents of XML documents, including
299 those which exploit the XML Namespace facility. XSDL documents have the '.xsd' file extension. See
300 [XML Schema Part 1: Structures](#).

301 **5 Symbols and Abbreviated Terms**

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

- 303 **5.1**
304 **GED**
305 Global Element Declaration

- 306 **5.2**
- 307 **MOF**
- 308 Managed Object Format
- 309 **5.3**
- 310 **URI**
- 311 Uniform Resource Identifier
- 312 **5.4**
- 313 **WSDL**
- 314 Web Services Description Language
- 315 **5.5**
- 316 **XML**
- 317 Extensible Mark-up Language
- 318 **5.6**
- 319 **XSD**
- 320 XML Schema Definition

321 **6 Namespace Prefixes and Schema Locations**

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

324 This specification defines namespaces as shown in Table 1.

325 **Table 1 – Namespaces**

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

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

328 **Table 2 – Namespace Prefixes**

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

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

330 **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

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

332 **Table 4 – XSD DSP Numbers**

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

333 7 Dereferencing Schema URI Locations in Order to Access XML 334 Schema

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

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

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

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

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

347 EXAMPLE:

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

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

354 EXAMPLE:

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

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

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

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

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

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

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

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

367 7.1.1 Validation of CIM Instances

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

373 DMTF permits the structure of the class to change in backwards-compatible ways within a release of a
374 major version of CIM. [DSP0004](#) (see “Schema Versions”) describes the nature of permitted changes in

375 this case. However, the changes permitted could cause XML schema validation errors. Implementations
 376 have a choice on how to be resilient to the changes permitted in such cases.

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

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

385 8 Mapping Primitive Datatypes

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

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

391 **Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes**

CIM Datatype	Corresponding Base XSD Datatypes	WS-CIM Datatypes
uint8	xs:unsignedByte	cim:cimUnsignedByte
sint8	xs:byte	cim:cimByte
uint16	xs:unsignedShort	cim:cimUnsignedShort
sint16	xs:short	cim:cimShort
uint32	xs:unsignedInt	cim:cimUnsignedInt
sint32	xs:int	cim:cimInt
uint64	xs:unsignedLong	cim:cimUnsignedLong
sint64	xs:long	cim:cimLong
string	xs:string	cim:cimString
boolean	xs:boolean	cim:cimBoolean
real32	xs:float	cim:cimFloat
real64	xs:double	cim:cimDouble
datetime	xs:duration xs:date xs:time xs:dateTime xs:string Depending on use-case See 8.1.	cim:cimDateTime
char16	xs:string With maxLength restriction = 1	cim:cimChar16
<class> REF	N/A	cim:cimReference See 8.2.

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

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

- 395 Octetstring
- 396 EmbeddedInstance
- 397 EmbeddedObject

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

400 8.1 cimDateTime Datatype

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

```

402 <xs:complexType name="cimDateTime">
403   <xs:choice>
404     <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
405     <xs:element name="Interval" type="xs:duration"/>
406     <xs:element name="Date" type="xs:date"/>
407     <xs:element name="Time" type="xs:time"/>
408     <xs:element name="Datetime" type="xs:dateTime"/>
409   </xs:choice>
410 <xs:anyAttribute namespace="##any" processContents="lax"/>
411 </xs:complexType>
    
```

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

413 **Table 6 – Rules for Converting datetime to cimDateTime**

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

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

416 **Table 7 – Rules for Converting cimDateTime to datetime**

cim:cimDateTime Element	Representation of datetime 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 0000000223****.*****:00
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>

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

417 8.2 CIM References

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

```
419 <xs:complexType name="cimReference">
420   <xs:sequence>
421     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
422   </xs:sequence>
423 <xs:anyAttribute namespace="##any" processContents="lax"/>
424 </xs:complexType>
```

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

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

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

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

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

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

```
447 <AssociatedComponent
448   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
449   <wsa:Address> . . . </wsa:Address>
450   . . . <!-- Other EPR elements as defined in the 2004/08 specification -->
451 </AssociatedComponent>
```


452 8.3 cimAnySimpleType Datatype

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

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

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

472 8.4 Derivation by Restriction

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

```
475 . . .
476 <this:Name xmlns:AdditionalAttribute=". . ." >
477     myName
478 </this:Name>
479 . . .
```

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

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

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

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

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

491 8.5 WS-CIM Canonical Values

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

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

498 9 CIM Class to XML Schema Mappings

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

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

504 9.1 Class Namespace

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

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

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

509 ['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-
510 version '/' cim-class-schema '_' cim-class-name)

511 Where:

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

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

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

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

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

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

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

525 9.2 Properties

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

529 9.2.1 General Principles

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

532 9.2.1.1 General Rules

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

- 534 • Every property of a CIM class shall be represented by a global element definition. Note that this
535 rule applies to properties locally defined on the class itself and properties inherited from
536 superclasses (see 9.4). The GED that corresponds to a CIM property shall exhibit the following
537 features:
 - 538 – The GED shall be defined in the namespace of the class in the XML Schema (see 9.1).
 - 539 – The name of the GED shall be the same as the name of the CIM property.
 - 540 – The type of the GED shall comply with the datatype conversion table defined in clause 8.
541 However, in some cases, depending on what qualifiers apply to the CIM property, it is
542 necessary to restrict the default type of the GED element. The complete specification of the
543 type of the GED shall comply with the normative rules for mapping qualifiers. (See 11.2 for
544 the normative rules for mapping specific qualifiers to XSD structures.)

- 545 • The GED of properties that are not arrays shall be specified with `nillable="true"`, if the
546 CIM property has a `Key` or `Required` qualifier with an effective value of `false`. The GED of
547 properties that are not arrays shall be specified without the `nillable` attribute (the default of
548 `nillable` is `false`), if the CIM property has a `Key` or `Required` qualifier with an effective
549 value of `true`. (See 11.3 for rules regarding the inheritance of qualifiers.)

550 NOTE: These rules do not apply to array properties. All GEDs that represent array properties shall be
551 specified with `nillable="true"`.

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

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

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

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

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

```
572 class EX_BaseComponent {
573     datetime InstallDate;
574     [ ...
575         Required, MaxLen ( 1024 ) ]
576     string Name;
577     string StatusDescriptions[];
578     string HealthStatus;
579 };
```

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

```
582 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true" />
583 <xs:element name="Name">
584   <xs:complexType>
585     <xs:simpleContent>
586       <xs:restriction base="cim:cimString">
587         <xs:maxLength value="1024" />
588         <xs:anyAttribute namespace="##any" processContents="lax" />
589       </xs:restriction>
590     </xs:simpleContent>
591   </xs:complexType>
592 </xs:element>
```

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

595 9.2.1.2 Runtime Rules for Attribute Value Assignment

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

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

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

```
606 class EX_SomeClass {
607   ...
608   [... Key]
609   string Name;
610 }
```

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

```
612 <EX_SomeClass>
613   ...
614   <Name cim:Key="true">MyName</Name>
615   ...
616 </EX_SomeClass>
```

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

618 9.2.2 Array Properties

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

620 9.2.2.1 General Rules

621 The rule for representing arrays is as follows:

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

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

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

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

```
631 [ ...
632     ArrayType ( "Indexed" ) ]
633 string StatusDescriptions[];
```

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

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

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

```
638 <EX_BaseComponent>
639   ...
640   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
641   <StatusDescriptions>AnotherStatusDescription</StatusDescriptions>
642   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
643   ...
644 </EX_BaseComponent>
```

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

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

```
648 [ ...
649     MaxLen ( 64 ) ]
650 string StatusDescriptions[];
```

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

```
654 <xs:element name="StatusDescriptions" nillable="true">
655   <xs:complexType>
656     <xs:restriction base="cim:cimString">
657       <xs:maxLength value="64"/>
658       <xs:anyAttribute namespace="##any" processContents="lax"/>
659     </xs:restriction>
660   </xs:complexType>
661 </xs:element>
```

662 9.2.2.2 Runtime Rules for Arrays

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

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

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

```
673 <EX_BaseComponent>
674   ...
675   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
676   <StatusDescriptions xsi:nil="true" />
677   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
678   ...
679 </EX_BaseComponent>
```

680 9.2.3 Properties with a ValueMap Qualifier

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

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

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

- 687 • If the `ValueMap` consists of only discrete values, the `ValueMap` shall be mapped to a
688 `<xs:restriction>` consisting of an enumeration as follows:
 - 689 – The base type of the restriction shall be the WS-CIM datatype corresponding to the CIM
690 datatype of the property (see 8).
 - 691 – Each discrete value of the `ValueMap` shall be mapped to a corresponding
692 `<xs:enumeration>` element within the restriction.
- 693 • If the `ValueMap` contains a value specifying a range, the whole `ValueMap` shall be mapped to a
694 `<xs:restriction>` consisting of a `<xs:union>` as follows:
 - 695 – The base type of the restriction shall be `cim:cimAnySimpleType` (see 8.3).
 - 696 – The elements of the `<xs:union>` shall be determined according to the following rules:
 - 697 • Discrete values shall be mapped to elements to an `<xs:restriction>` as
698 described in the first rule with the exception that the base type of the restriction shall
699 be the corresponding base XSD type of the CIM datatype of the property (see
700 clause 8);
 - 701 • Bounded ranges (*m..n*) shall be mapped to an `<xs:restriction>` consisting of
702 `<xs:minInclusive="m">` and `<xs:maxInclusive="n">` where the restriction
703 base type shall be the corresponding base XSD type of the CIM datatype of the
704 property;
 - 705 • Unbounded ranges open on the left (*...n*) shall be mapped to an
706 `<xs:restriction>` consisting of `<xs:maxInclusive="n">` where the restriction
707 base type shall be the corresponding base XSD type of the CIM datatype of the
708 property;
 - 709 • Unbounded ranges open on the right (*m...*) shall be mapped to an
710 `<xs:restriction>` consisting of `<xs:minInclusive="m">` where the restriction
711 base type shall be the corresponding base XSD type of the CIM datatype of the
712 property;
 - 713 • Open ranges (*..*) shall be mapped to an `<xs:union>` consisting of all discrete values
714 and/or ranges that are unclaimed by the other values and ranges in the `ValueMap` by
715 applying the preceding rules for constructing the elements of the `<xs:union>`
716 recursively.

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

```
718 [ ...
719     ValueMap { "OK", "Error", "Unknown" } ]
720 string HealthStatus;
```

721 The HealthStatus property is therefore mapped as follows:

```
722 <xs:element name="HealthStatus" nillable="true">
723     <xs:complexType>
724         <xs:simpleContent>
725             <xs:restriction base="cim:cimString">
726                 <xs:enumeration value="OK"/>
727                 <xs:enumeration value="Error"/>
728                 <xs:enumeration value="Unknown"/>
729                 <xs:anyAttribute namespace="##any" processContents="lax"/>
730             </xs:restriction>
731         </xs:simpleContent>
732     </xs:complexType>
733 </xs:element>
```

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

```
735 [ ...
736 ValueMap { "0", "1", "2", "3..15999", "16000..65535" },
737     Values { "Unknown", "Other", "Not Applicable", "DMTF Reserved",
738             "Vendor Reserved" }}
739 uint16 PortType;
```

740 The PortType property is therefore mapped as follows:

```
741 <xs:element name="PortType" nillable="true">
742     <xs:complexType>
743         <xs:simpleContent>
744             <xs:restriction base="cim:cimAnySimpleType">
745                 <xs:simpleType>
746                     <xs:union>
747                         <xs:simpleType>
748                             <xs:restriction base="xs:unsignedShort">
749                                 <xs:enumeration value="0"/>
750                                 <xs:enumeration value="1"/>
751                                 <xs:enumeration value="2"/>
752                             </xs:restriction>
753                         </xs:simpleType>
754                         <xs:simpleType>
755                             <xs:restriction base="xs:unsignedShort">
756                                 <xs:minInclusive value="3"/>
757                                 <xs:maxInclusive value="15999"/>
758                             </xs:restriction>
759                         </xs:simpleType>
760                         <xs:simpleType>
761                             <xs:restriction base="xs:unsignedShort">
762                                 <xs:minInclusive value="16000"/>
763                                 <xs:maxInclusive value="65535"/>
764                             </xs:restriction>
765                         </xs:simpleType>
766                     </xs:union>
767                 </xs:simpleType>
768             <xs:anyAttribute namespace="##any" processContents="lax"/>
769         </xs:simpleContent>
770     </xs:complexType>
771 </xs:element>
```

```

769     </xs:restriction>
770   </xs:simpleContent>
771 </xs:complexType>
772 </xs:element>

```

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

```

775 [ ...
776 ValueMap { "1", "2", "3", "4", "5", "6", "7", "..", "16000.." },
777   Values { "Other", "Create", "Delete", "Detect", "Read", "Write",
778     "Execute", "DMTF Reserved", "Vendor Reserved" } ]
779 uint16 Activities;

```

780 The Activities property is therefore mapped as follows:

```

781 <xs:element name="Activities" nillable="true">
782   <xs:complexType>
783     <xs:simpleContent>
784       <xs:restriction base="cim:cimAnySimpleType">
785         <xs:simpleType>
786           <xs:union>
787             <xs:simpleType>
788               <xs:restriction base="xs:unsignedShort">
789                 <xs:enumeration value="1"/>
790                 <xs:enumeration value="2"/>
791                 <xs:enumeration value="3"/>
792                 <xs:enumeration value="4"/>
793                 <xs:enumeration value="5"/>
794                 <xs:enumeration value="6"/>
795                 <xs:enumeration value="7"/>
796               </xs:restriction>
797             </xs:simpleType>
798             <xs:simpleType>
799               <xs:union>
800                 <xs:simpleType>
801                   <xs:restriction base="xs:unsignedShort">
802                     <xs:enumeration value="0"/>
803                   </xs:restriction>
804                 </xs:simpleType>
805                 <xs:simpleType>
806                   <xs:restriction base="xs:unsignedShort">
807                     <xs:minInclusive value="8"/>
808                     <xs:maxInclusive value="15999"/>
809                   </xs:restriction>
810                 </xs:simpleType>
811               </xs:union>
812             </xs:simpleType>
813             <xs:simpleType>
814               <xs:restriction base="xs:unsignedShort">
815                 <xs:minInclusive value="16000"/>
816               </xs:restriction>
817             </xs:simpleType>
818           </xs:union>
819         </xs:simpleType>
820       <xs:anyAttribute namespace="##any" processContents="lax"/>
821     </xs:restriction>
822   </xs:simpleContent>
823 </xs:complexType>
824 </xs:element>

```


825 9.2.4 Octetstring Properties

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

829 9.2.4.1 General Rules

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

- 831 • A `uint8` array that is designated as an octetstring shall be mapped to a single XSD element of
832 the type `cim:cimBase64Binary`. The rules for mapping properties defined in 9.2.1.1 apply to
833 this mapping.
- 834 • A `string` array that is designated as an octetstring shall be mapped to an array of type
835 `cim:cimHexBinary`. The rules for mapping arrays defined in 9.2.2.1 apply to this mapping.
- 836 • The XML payload does not include the 0x or the length bytes for the string form of octet string.
837 The `uint8` form does not include the length bytes.

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

```
839 [...
840     Description ("The DER-encoded raw public key. " ),
841     OctetString ]
842 uint8 PublicKey[];
```

843 It would be represented by the following XSD:

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

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

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

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

```
848 [...
849     Description ("A CRL, or CertificateRevocationList, is a list of certificates which the "
850               "CertificateAuthority has revoked and which are not yet expired. " ),
851     Octetstring]
852 string CRL[];
```

853 It would be represented by the following XSD:

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

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

```
856 <CRL>76C8A4...</CRL>
857 <CRL>75D4E1...</CRL>
858 <CRL>B1C335...</CRL>
```

859 9.2.4.2 Runtime Value Conversion Rules

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

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

- 866 • A `uint8` array that is designated as an octetstring shall be converted to its corresponding
867 representation in `base64Binary` such that the ordered set of array elements is concatenated into
868 a binary multi-octet string, which is converted to base64 encoding. This encoding represents the

869 base64Binary value. The order of the unsigned 8-bit integer array shall be preserved when
870 mapped to the characters of the XML value.

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

874 9.2.5 EmbeddedObject and EmbeddedInstance Properties

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

879 9.2.5.1 General Rules

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

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

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

```
886 [ ...
887     EmbeddedObject ( "...") ]
888 string TheObject;
```

889 It would have the following XSD representation:

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

891 9.2.5.2 Runtime Value Conversion Rules

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

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

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

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

```
907 class CIM_Part {
908     string Label;
909     int PartNo;
910 };
911 class CIM_Component {
```

```

912 [Key]
913 string ID;
914 [EmbeddedInstance("CIM_Part")]
915 string Part;
916 };

```

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

```

919 <CIM_Component xmlns="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Component"
920 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
921   <ID>ua123</ID>
922   <Part xmlns:e="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Part"
923     xsi:type="e:CIM_Part_Type">
924     <e:Label>Front Panel</e:Label>
925     <e:PartNo>19932</e:PartNo>
926   </Part>
927 </CIM_Component>

```

928 9.3 Class Structure

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

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

933 9.3.1 General Rules

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

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

```
942 '<x:element ref=' QName ' . ' Attributes '/>'

```

943 Where:

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

946 Elements that belong to the class complex type should be ordered by Unicode code point
 947 (binary) order of their CIM property name identifiers. (See ANNEX C for information about
 948 collation.)

- 949 • In existing service implementations, the collation sequence of property name
 950 identifiers should be in Unicode code point (binary) order.
- 951 • In existing service implementations, the collation sequence of property name
 952 identifiers may be according to the Unicode Collation Algorithm (UCA) with its default
 953 settings. This algorithm is deprecated and should not be used in future service
 954 implementations.

- 955 • In DMTF XML schema representations of CIM classes, and in new service
956 implementations, the collation sequence of property name identifiers shall be in code
957 point (binary) order.

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

- 959 • All elements that do not represent array properties shall have `minOccurs="0"` except for
960 elements that correspond to properties that are designated with a `Key` or `Required` qualifier
961 with an effective value of `true`. Elements that do not represent arrays and represent key and
962 required properties shall have `minOccurs="1"`. Because 1 is the default value for
963 `minOccurs`, it does not need to be explicitly expressed.
 - 964 • All elements that represent array properties shall have `minOccurs="0"`. If the array size is
965 specified in the CIM definition, the array property shall have `maxOccurs="array size"`. If
966 the array size is not specified, the array property shall have `maxOccurs="unbounded"`.
 - 967 • All elements except arrays shall have `maxOccurs="1"`. Because 1 is the default value for
968 `maxOccurs`, it does not need to be explicitly expressed.
 - 969 • Array properties (see 9.2.1.1) shall have a multiplicity that corresponds to the specification of
970 their size in CIM. A bounded array in CIM shall be specified with a `maxOccurs` equal to the size
971 of the array. If no size is specified in CIM, the schema element shall be specified with
972 `minOccurs="0"` and `maxOccurs="unbounded"`.
 - 973 • The schema of the CIM class shall support an open schema. Open schema means different
974 protocols are able to add protocol-specific elements to instance documents.
 - 975 – To allow Open Content, the final element in the sequence shall be as follows:
- ```
976 <xs:any namespace="##other" processContents="lax" minOccurs="0"
977 maxOccurs="unbounded" />
```
- 978           – To allow attributes to be added to the element that represents the CIM class, following the  
979 sequence, the complex type shall allow any attribute to be added to the class with an  
980 `xs:anyAttribute` element, as follows:
- ```
981           <xs:anyAttribute namespace="##any" processContent="lax" />
```
- 982 • The class itself shall be represented by a GED of the type defined in the preceding rule. The
983 name of this element shall be the name of the CIM class including its CIM schema name.

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

```
985 <xs:complexType name="EX_BaseComponent_Type">
986   <xs:sequence>
987     <xs:element ref="class:HealthStatus" minOccurs="0" />
988     <xs:element ref="class:InstallDate" minOccurs="0" />
989     <xs:element ref="class:Name" />
990     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded" />
991     <xs:any namespace="##other" processContents="lax" minOccurs="0"
992       maxOccurs="unbounded" />
993   </xs:sequence>
994   <xs:anyAttribute namespace="##any" processContent="lax" />
995 </xs:complexType>
996 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type" />
```

997 The complete mapping of this class is provided in B.2.1.

998 9.3.2 Runtime Property Value Rules

999 Runtime inclusion of property values for instance documents based on the XML Schema for CIM classes
1000 is defined by the following rules:

- 1001 • For "get" operations, CIM service implementations returning the class's GEDs may omit
1002 schema-optional (`minOccurs="0"`) properties that have NULL values from responses. Clients
1003 are to interpret such omitted properties as having NULL values for those properties.
- 1004 • Empty arrays (arrays with no members) shall not be included in responses to "get" requests. If
1005 the array is required, clients shall interpret the absence of all elements for the array to mean
1006 that the array is empty (no members). If the array is not required, clients shall interpret the
1007 absence of all elements for the array to mean that the array is either empty or NULL.
- 1008 • A WS-CIM server shall return all current elements of an array.
- 1009 • For "set" operations using the class's GEDs, clients may omit schema-optional
1010 (`minOccurs="0"`) properties. Service interpretation of the absence of such properties is
1011 protocol dependent.
- 1012 • An instance document conformant to this specification shall include any elements from a foreign
1013 namespace at the end of the sorted list of CIM class elements. Elements from a foreign
1014 namespace may appear in any order.
- 1015 • If a `Version` qualifier is associated with the CIM class, the `cim:Version` attribute may be
1016 applied to the element that represents the class in an XML instance document. Use of the
1017 `cim:Version` attribute shall conform to the following rule:

1018 If the attribute is present, its value shall be assigned as the value of the `Version` qualifier that
1019 is associated with the CIM class, in the XML instance. If the CIM class does not have the
1020 `Version` qualifier associated with it, this attribute shall be omitted.

1021 The MOF typically contains the `Version` qualifier, which specifies the latest version of the CIM class in
1022 CIM:

```
1023 [ ... Version ( "2.10.2" ) ]
1024 class EX_SomeClass {
1025     ...
1026 }
```

1027 The class may be represented in an instance document as follows:

```
1028 <EX_SomeClass cim:Version="2.10.2">
1029     ...
1030 </EX_SomeClass>
```

1031 9.4 Class Inheritance

1032 CIM inheritance is not modeled in the XML Schema of classes or within XML instances.

1033 Class inheritance is governed by the following rules:

- 1034 • Besides including the GEDs for the properties defined in a class (see 9.2.1.1), the namespace
1035 for a class shall also include the GEDs for properties inherited from its superclasses. The class
1036 type definition shall contain references to GEDs for all properties defined in and inherited into
1037 the class according the rules in 9.3.1.
- 1038 • In general, classes inherit all properties specified in or inherited by their superclasses along with
1039 all qualifiers that are specified as `ToSubClass`. However, properties with the same name may

1040 be encountered within an inheritance chain. The `Override` qualifier determines special
 1041 behaviors that shall be observed by conversion algorithms when encountering properties with
 1042 duplicate names in the inheritance chain. The following rules govern the mapping of properties
 1043 with duplicate names:

1044 – When multiple properties in the inheritance chain that have the same name are not
 1045 overridden (that is, the effective value of the `Override` qualifier throughout the inheritance
 1046 chain is `NULL`), the property and its qualifiers in the most derived class shall be mapped.
 1047 All other duplicate named properties shall not be mapped.

1048 – When a property in a derived class overrides another property (of the same name and
 1049 type) in a superclass, the property in the most derived class shall be mapped, including all
 1050 qualifiers inherited from the overridden property. The overridden property shall not be
 1051 mapped.

1052 • The inheritance of qualifiers that pertain to properties shall comply with the inheritance rules
 1053 regarding qualifiers in B.3.

1054 • The definition of a derived class shall follow all other rules for constructing classes as defined in
 1055 9.3.1.

1056 NOTE: The metadata fragments for a property shall include any inherited qualifiers, subject to the qualifier inheritance
 1057 rules defined in 11.3. For more information about metadata fragments, see clause 11.

1058 Inheritance rests on the same type of examples presented in 9.2 and 9.3. The only addition is that the
 1059 properties inherited from a class's superclasses are included in the GEDs and class complex type
 1060 definition. For a complete example of inheritance, see B.2.2.

1061 9.5 Method Parameters

1062 The invocation of a CIM extrinsic method is represented by two separate messages:

- 1063 • the request input message, which represents the invocation of the method and the set of input
 1064 parameters
- 1065 • the response output message, which represents the output parameters and the method return
 1066 value in the successful case

1067 This clause specifies the XML Schema for these elements. These elements are then included as parts in
 1068 the WSDL input and output messages (see 10.1).

1069 9.5.1 General Rules

1070 The rules in this clause apply to mapping method input and output parameters and method return values.

1071 The GED used for the request input message is called the *input message GED*. The GED used for the
 1072 response output message is called the *output message GED*. The following rules specify the definition of
 1073 these GEDs:

1074 • The class namespace of the CIM class being mapped shall contain the input and output
 1075 message GEDs of all methods owned by the class and inherited from the superclasses. See
 1076 9.5.2, which defines class ownership of methods inherited from superclasses.

1077 • The names of these GEDs are defined by the following rules:

1078 – The `name` of the input message GED shall be the name of the CIM method with `_INPUT`
 1079 appended.

1080 – The `name` of the output message GED shall be the name of the CIM method with `_OUTPUT`
 1081 appended.

- 1082 • Each GED shall be defined as a complex type containing an `xs:sequence` of in-line elements
 1083 that represent the respective input or output parameters and return value of the CIM method as
 1084 immediate children. This structure is further defined in the rest of this clause.

1085 The following rules define the mapping of individual input and output parameters and the return value of
 1086 the CIM method, and the structure of the complex type used for defining the GEDs:

- 1087 • Input and output parameters of a CIM method shall be mapped to elements with the same
 1088 name as the corresponding parameter name. The following rules define the features of these
 1089 elements:
- 1090 – The type of an element that represents an input or output parameter shall be mapped as
 1091 defined in clause 8.
 - 1092 – Parameters that are not qualified with a `Required` qualifier shall be mapped to elements
 1093 that contain the `nillable="true"` attribute.
 - 1094 • The complex type used to define the type of the input message GED shall contain all and only
 1095 those elements that correspond to method parameters that have their `In` qualifier effectively
 1096 defined as `true`. The sequence of these elements in the complex type shall correspond to the
 1097 sequence of the input parameters in the CIM definition of the method.
- 1098 If the method has no input parameters, the complex type used in the GED shall be empty (that
 1099 is, `<xs:complexType/>`).
- 1100 • The complex type used to define the type of the output message GED shall contain all elements
 1101 that correspond to method parameters that have their `Out` qualifier effectively defined as `true`.
 1102 The sequence of these elements in the complex type shall correspond to the sequence of the
 1103 output parameters in the CIM definition of the method.
 - 1104 • The complex type used to define the output message GED shall contain an element of
 1105 name="ReturnValue" as the final element in its sequence. The following rules govern the
 1106 structure of this element:
 - 1107 – The XSD type of this element shall be mapped from the CIM method type in compliance
 1108 with the datatype conversion defined in clause 8.
 - 1109 – The element shall include the attribute `nillable="true"`. (See the following note.)
 - 1110 • Parameters and return values may be defined in CIM with `ValueMap` qualifiers. Mapping these
 1111 `ValueMaps` to metadata fragments is required (see 11.1). In addition, a `ValueMap` shall be
 1112 mapped to an enumeration/union associated with the mapped parameter or return value in the
 1113 XSD (see 11.2). The mapping shall conform to the rules for mapping `ValueMaps` described in
 1114 9.2.3.

1115 Notes on the preceding rules:

- 1116 • A parameter shall occur in both the complex types used to define the input and output message
 1117 GEDs if it has both the `In` and `Out` qualifiers effectively defined as `true`.

1118 [DSP0004](#) allows NULL as the default return value for all methods. Thus, this specification must allow for
 1119 the possibility that the return value of any method may be NULL.

1120 9.5.2 Inheritance of Methods

1121 Classes may inherit some of the methods that they own. In general, classes inherit all methods specified
 1122 in or inherited by their superclasses, along with all qualifiers that are specified as `ToSubClass`. The
 1123 `Override` qualifier, however, determines special behavioral considerations on the part of conversion
 1124 algorithms. The following rules govern the inheritance of methods under conditions of override:

- 1125 • When multiple methods in the inheritance chain that have the same name are not overridden,
 1126 the method in the most derived class shall be mapped. Any other duplicate, but not overridden,
 1127 methods shall not be mapped.
- 1128 • When a method in a derived class overrides another method (of the same name and signature)
 1129 in a superclass, the method in the most derived class shall be mapped, including all qualifiers
 1130 inherited from the overridden methods. The overridden method shall not be mapped.
- 1131 • The inheritance of qualifiers pertaining to methods shall comply with the inheritance rules
 1132 regarding qualifiers in B.3.

1133 EXAMPLE: As an example of the preceding rules, consider the following MOF method definition extracted from the
 1134 example in B.1.2. (See B.2.2 for the complete example.)

```
1135 class EX_DerivedComponent
1136 {
1137 ...
1138 uint32 RequestStateChange([IN] uint16 RequestedState, [OUT] [IN(False)] CIM_SomeClass REF
1139 ResultClass, [IN] datetime TimeoutPeriod);
1140 };
```

1141 The input parameters, designated in the MOF by the `In` qualifier, would be mapped as follows:

```
1142 <xs:element name="RequestStateChange_INPUT">
1143 <xs:complexType>
1144 <xs:sequence>
1145 <xs:element name="RequestedState" type="cim:cimUnsignedShort"
1146 nillable="true"/>
1147 <xs:element name="TimeoutPeriod" type="cim:cimDateTime"
1148 nillable="true"/>
1149 </xs:sequence>
1150 </xs:complexType>
1151 </xs:element>
```

1152 The output parameters, designated in the MOF by the `Out` qualifier, would be mapped in the following
 1153 way. Note that the complex type includes an element that represents the return value of the CIM method.

```
1154 <xs:element name="RequestStateChange_OUTPUT">
1155 <xs:complexType>
1156 <xs:sequence>
1157 <xs:element name="ResultClass" type="cim:cimReference"
1158 nillable="true"/>
1159 <xs:element name="ReturnValue" type="cim:cimUnsignedInt"
1160 nillable="true"/>
1161 </xs:sequence>
1162 </xs:complexType>
1163 </xs:element>
```

1164 9.5.3 Parameters and Return Values with ValueMaps

1165 Input and output parameters and return values of the method may be specified with a `ValueMap` qualifier.
 1166 The `ValueMap` shall be mapped to the XSD element that represents the parameter or return value.

1167 The RequestedState input parameter must be defined as an enumeration in accordance with its
1168 ValueMap (see B.1.2 for this ValueMap):

```
1169 <xs:element name="RequestedState" nillable="true">
1170   <xs:complexType>
1171     <xs:simpleContent>
1172       <xs:restriction base="cim:cimUnsignedShort">
1173         <xs:enumeration value="2"/>
1174         <xs:enumeration value="3"/>
1175         <xs:enumeration value="4"/>
1176         <xs:anyAttribute namespace="##any" processContents="lax"/>
1177       </xs:restriction>
1178     </xs:simpleContent>
1179   </xs:complexType>
1180 </xs:element>
```

1181 9.6 CIM Instances

1182 This clause describes the representation of CIM instances. The intended scope is all representations of
1183 CIM instances used in any protocols.

1184 CIM instances are represented according to the following rules:

- 1185 • Representations of CIM instances shall be XML instance documents that conform to the XSD
1186 schema for their CIM creation class, as defined in clause 9.
- 1187 • The class namespace used within an instance document shall be a namespace URI and it shall
1188 be defined as follows:

1189 ['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-
1190 version '/' cim-class-schema '_' cim-class-name)

1191 Where:

- 1192 – [wscim-major-version](#) is the major version number of this specification. Note that this
1193 version number changes only if there are incompatible changes in the specification.
- 1194 – [cim-schema-major-version](#) is the major version number of the CIM schema version to
1195 which the class being converted belongs. Note that this version number changes only if
1196 there are incompatible changes in the CIM schema.
- 1197 • The location of the specific schema used to construct the instance should be declared by use of
1198 the xsi:schemaLocation attribute where the namespace URI and the specific class schema
1199 document URI location are combined as the value of the attribute, separated by whitespace.
1200 This provides a means for a recipient of the instance to determine which version of CIM defines
1201 the structure of this instance.

1202 For example:

1203 If the instance document is constructed under CIM schema version 2.11.0 and WS-CIM
1204 mapping specification 1.3.0, the following xsi:schemaLocation attribute should be specified in
1205 the instance document (where *ClassName* is the name of the CIM class of the instance):

```
1206   xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/ClassName  
1207   http://schemas.dmtf.org/wbem/wscim/1.3.0/cim-schema/2.11.0/ClassName.xsd"
```

1208 If the URI in the attribute value can be de-referenced, then strict XML schema validation can be achieved.

1209 9.7 Superclass Class Representation

1210 The CIM Schema defines CIM classes in subclass / superclass relationships (hierarchies). For example,
 1211 MOF reflects this structure in the definition of the class. A MOF statement of the following form defines
 1212 CIM_ClassA as a subclass of the superclass CIM_ClassB:

```
1213 class CIM_ClassA : CIM_ClassB
```

1214 A MOF statement of the following form defines CIM_ClassC as a top-level class with no superclass:

```
1215 class CIM_ClassC
```

1216 Some management protocols may require a representation of the subclass/superclass hierarchy of a
 1217 class as a value of an XML element in instance documents. The XSD mechanism by which to support
 1218 representing this structure as a value of an XML element is provided in a separate schema that may be
 1219 imported by protocols that require this capability for their instance documents.

1220 The immediate subclass/superclass relationship of a class shall be mapped to a single GED in the
 1221 classhierarchy namespace according to the following rules:

- 1222 • Elements that describe the subclass / superclass relationships shall be placed in a separate
 1223 schema from the WS-CIM class schema. For subclass / superclass relationships defined in the
 1224 CIM Schema, the schema shall be named as follows:

```
1225     'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-  

  1226     major-version '/classhierarchy'
```

1227 Where:

- 1228 – [wscim-major-version](#) is the major version number of this specification. Note that this
 1229 version number changes only if there are incompatible changes in the specification.
- 1230 – [cim-schema-major-version](#) is the major version number of the CIM schema version to
 1231 which the class being converted belongs. Note that this version number changes only if
 1232 there are incompatible changes in the CIM schema.

1233 This schema shall import the <http://schemas.dmtf.org/wbem/wscim/n/classhiertype> namespace, where 'n'
 1234 represents the version number of this namespace.

1235 The class being mapped shall be represented by a GED whose name is derived from the name of the
 1236 CIM class, appended with "_Class". This element shall be defined by an anonymous complex type that
 1237 follows one of the following patterns:

- 1238 • The WS-CIM schema of a top-level class, XXX_ClassC, shall define the _Class element as a
 1239 restriction of the `ctype:ClassHierarchyType`. The following pattern illustrates this rule:

```
1240 <xs:element name="XXX_ClassC_Class">  

  1241   <xs:complexType>  

  1242     <xs:complexContent>  

  1243       <xs:restriction base="ctype:ClassHierarchyType" />  

  1244     </xs:complexContent>  

  1245   </xs:complexType>  

  1246 </xs:element>
```

- 1247 • The WS-CIM schema of a subclass, XXX_ClassA, that is derived by a superclass,
 1248 XXX_ClassB, shall restrict the `ctype:classHierarchyType` to contain a single element that
 1249 references the corresponding _Class GED of the superclass. The following pattern illustrates
 1250 this rule:

```

1251 <xs:element name="XXX_ClassA_Class">
1252     <xs:complexType>
1253         <xs:complexContent>
1254             <xs:restriction base="ctype:ClassHierarchyType">
1255                 <xs:sequence>
1256                     <xs:element ref="chier:XXX_ClassB_Class" />
1257                 </xs:sequence>
1258             </xs:restriction>
1259         </xs:complexContent>
1260     </xs:complexType>
1261 </xs:element>

```

1262 See B.2.4 for an example classhierarchy schema.

1263 10 CIM Methods to WSDL Mappings

1264 This clause defines the structures that are necessary to define the messages and operation structures
 1265 required for mapping a CIM method to WSDL.

1266 10.1 Defining WSDL Message Structures

1267 This clause provides the rules for creating WSDL message structures.

1268 The rules that govern the creation of WSDL message structures for a method are as follows:

- 1269 • The name of the WSDL input message should be the name of the CIM method plus
 1270 `_InputMessage`. The following rules specify the structure of this element:
 - 1271 – The name of the `wsdl:part` element should be "body".
 - 1272 – The `element` attribute of the `wsdl:part` element shall specify the QName of the input
 1273 message GED for the CIM method (see 9.5).
- 1274 • The name of the WSDL output message should be the name of the CIM method plus
 1275 `_OutputMessage`. The following rules specify the structure of this element:
 - 1276 – The name of the `wsdl:part` element should be "body".
 - 1277 – The `element` attribute of the `wsdl:part` element shall specify the QName of the output
 1278 message GED defined for the CIM method (see 9.5).

1279 **EXAMPLE:** The `wsdl:message` elements for the `RequestStateChange` CIM method (see 9.5) would
 1280 be specified in the WSDL document as follows. The `wsdl:message` intended for input to the WSDL
 1281 operation would be as follows (where the "class:" namespace prefix represents the namespace of the
 1282 class whose interface is being exposed through this WSDL):

```

1283 <wsdl:message name="RequestStateChange_InputMessage">
1284     <wsdl:part name="body"
1285         element="class:RequestStateChange_INPUT" />
1286 </wsdl:message>

```

1287 See B.3 for an example that shows the complete specification of the WSDL messages for this operation.

1288 10.2 Defining WSDL Operation Structures

1289 This specification defines *only* the structure of WSDL `portType` operations.

1290 The rules governing the structure of the WSDL operations used to invoke CIM methods are as follows:

- 1291 • The name attribute of the `wsdl:operation` element shall be the name of the CIM method.
- 1292 • The name attributes of the `wsdl:input` and `wsdl:output` child elements should be the name
1293 of the `wsdl:messages` that are referenced by these elements.
- 1294 • The message attributes of the `wsdl:input` and `wsdl:output` elements shall specify the
1295 QName of input and output message elements defined in 10.1.

1296 EXAMPLE: The `RequestStateChange` CIM method (see 9.5) is defined as follows:

```
1297 <wsdl:definitions
1298   ...
1299   xmlns:thisWSDL="http://. . .wsdl"
1300   ...>
1301 <wsdl:operation name="RequestStateChange">
1302   <wsdl:input name="RequestStateChange_InputMessage"
1303     message="thisWSDL:RequestStateChange_InputMessage" />
1304   <wsdl:output name="RequestStateChange_OutputMessage"
1305     message="thisWSDL:RequestStateChange_OutputMessage" />
1306 </wsdl:operation>
1307 </wsdl:definitions>
```

1308 This definition should be included in the `wsdl:portType` section of a WSDL document of a service that
1309 makes the CIM `RequestStateChange` method available to clients.

1310 10.3 Defining `wsa:Actions`

1311 The Addressing specifications ([Web Services Addressing \(WS-Addressing\) 1.0 – Core](#) and [DSP0226](#),
1312 "Management Addressing" clause) define the information model and syntax for the abstract messaging
1313 property Action. This property is defined as an IRI ([RFC 3987](#)), which identifies the semantics implied by
1314 a message (input, output, or fault). For the purposes of this specification, the Action property is restricted
1315 to a URI ([RFC 3986](#)) (a sequence of characters from a limited subset of the repertoire of US-ASCII
1316 characters).

1317 The details of how the action URI is specified on description artifacts are left to the specific protocol-
1318 binding specifications. Action URIs for faults are always left to the protocol-binding specifications.

1319 The rules for constructing WSA action URIs for input and output operation elements are as follows:

- 1320 • The action URI for an input message shall have the following form:
1321 `class-namespace-name '/' input-name`
1322 Where:
1323 – `class-namespace-name` is the full namespace name of the class being mapped as defined
1324 in 9.1.
1325 – `input-name` is the name of the CIM method.
- 1326 • The action URI for an output message shall have the following form:
1327 `class-namespace-name '/' output-name 'Response'`

1328 Where:

- 1329 – class-namespace-name is the full namespace name of the class being mapped as defined
- 1330 in 9.1.
- 1331 – output-name is the name of the output CIM method.

1332 EXAMPLE: The action URI for the input message of the `RequestStateChange` method is as follows:

```
1333 wsa:Action="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/EX_DerivedComponent /
1334 RequestStateChange"
```

1335 The action URI for the output message of the `RequestStateChange` method is as follows:

```
1336 wsa:Action="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent /
1337 RequestStateChangeResponse"
```

1338 See B.3 for an example that shows a complete mapping of the `RequestStateChange` method.

1339 11 Qualifier Mappings

1340 This clause defines the mapping of qualifiers to metadata fragments. The definition of the container for
1341 metadata fragments is left to the specific protocols to specify.

1342 11.1 General Format

1343 Rules for mapping qualifiers fall into two categories:

- 1344 • rules that describe the type definitions of qualifiers
- 1345 • rules that describe the XSD elements to which qualifiers are mapped

1346 This specification provides type definitions for the qualifier types used in CIM in `common.xsd` and
1347 mappings for all CIM qualifiers in the `qualifiers.xsd` (Annex A.2). It is expected that user-defined qualifiers
1348 follow the mapping rules for mapping qualifiers described in the following clauses.

1349 In addition to the mapping rules, user-defined qualifier mappings are governed by the following rules:

- 1350 • User-defined qualifiers shall not use any qualifier namespace defined in this specification.

1351 The qualifier types (types with names of the qualifier *Type*) defined in `common.xsd` should be used to
1352 define user-defined qualifiers. In the majority of cases, it is not necessary to create a new type definition
1353 to define a qualifier.

1354 11.1.1 Single-Valued Qualifiers

1355 This clause describes the rules for mapping single-valued qualifiers.

1356 Single-valued qualifiers that have an effective value that matches their default value shall not be mapped
1357 to a metadata fragment.

1358 The rules for mapping single-valued qualifiers that have an effective value that is not their global default
1359 value are as follows:

- 1360 • The rules for defining the type of a single-valued qualifier are as follows:
 - 1361 – Single-valued qualifiers shall be mapped to a complex type that is an extension of a simple
 - 1362 content.
 - 1363 – The base type of this complex type shall correspond to the type of qualifier according to the
 - 1364 datatype conversion rules defined in clause 8.

- 1365 – The complex type shall extend the base type with a Boolean attribute of the name
1366 *qualifier*. This attribute is defined in the *cim* namespace. This attribute shall be
1367 specified with the XML Schema attribute `use="required"`.
- 1368 • The rules for mapping a single-valued qualifier are as follows:
- 1369 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of
1370 the qualifier and which has been defined by the preceding rule. The name of the element
1371 shall be the name of the qualifier.
- 1372 – The value of the `cim:qualifier` attribute in the mapped metadata fragment shall be `true`
1373 in all mappings.
- 1374 – Single-valued qualifiers implicitly have the multiplicity `minOccurs="0" maxOccurs="1"`.
1375 Protocols should provide a mechanism by which to express this multiplicity in the schema
1376 of the document that contains generated metadata fragments.
- 1377 • The format of a schema element for defining a single-valued qualifier is as follows:
- 1378

```
'<xs:element name=" " cim:qualifier-name ' " type=" " qualifier-type '"/>'
```
- 1379 Where:
- 1380 – `qualifier-type` is typically one of the *QualifierType* types defined in `common.xsd` (but may be
1381 a user-defined type that complies with the mapping rules).

1382 `cim:qualifier-name` is the name of the qualifier, qualified by its namespace prefix.

1383 EXAMPLE: A generalized example of the mapping of a single-valued qualifier is as follows:

```
1384 <ns:QualifierName cim:qualifier="true">
1385    value
1386 </ns:QualifierName>
```

1387 Where:

- 1388 – `QualifierName` is the name of the qualifier.
- 1389 – `ns` is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1390 – `value` is the string representation of the qualifier value.

1391 For example, the mapping of the CIM qualifier `Abstract` is as follows:

```
1392 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
```

1393 11.1.2 Multi-Valued Qualifiers

1394 This clause describes the rules for mapping multi-valued qualifiers.

1395 Multi-valued qualifiers are qualifiers that are arrays. Multi-valued qualifiers that have an effective value
1396 that matches their default value shall not be mapped to a metadata fragment.

1397 The rules for mapping multi-valued qualifiers that have an effective value that is not their global default
1398 value are as follows:

- 1399 • The rules for defining the type of a multi-valued qualifier are as follows:
- 1400 – Multi-valued qualifiers shall be mapped to a complex type that is an extension of a complex
1401 content.
- 1402 – The base type of this complex type shall correspond to the single-valued *QualifierType* of
1403 the member elements from which the qualifier array is constructed.

- 1404 • The rules for mapping a multi-valued qualifier are as follows:
 - 1405 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of
 - 1406 the qualifier and which has been defined by the preceding rule. The name of the element
 - 1407 shall be the same as that of the qualifier itself.
 - 1408 – The value of the `cim:qualifier` attribute in the metadata fragments shall be `true` in all
 - 1409 mappings.
 - 1410 – Multi-valued qualifiers implicitly have the default multiplicity `minOccurs="0"`
 - 1411 `maxOccurs="unbounded"`. Qualifier declarations may explicitly set different bounds on
 - 1412 an array qualifier. Protocols should provide a mechanism by which to express the size of
 - 1413 an array qualifier in the schema of the document that contains generated metadata
 - 1414 fragments. The `minOccurs` and `maxOccurs` values shall correspond either to the default
 - 1415 values if no qualifier array size is declared or to the declared qualifier array size.

1416 NOTE: The `common.xsd` file defines a string array of qualifier values, `qualifierSArray`. This definition complies
 1417 with the first mapping rule in this clause. Therefore, in the majority of cases, it is not necessary to create a new array
 1418 complex type definition to define a multi-valued qualifier. Rather, it is sufficient to use the `qualifierSArray` type
 1419 defined in `common.xsd`.

1420 The format for a schema for defining a multi-valued qualifier is as follows:

```
1421 <xs:element name="" cim:qualifier-name "" type="" qualifier-array-type ""/>'
```

1422 Where:

- 1423 – `cim:qualifier-name` is the name of the qualifier, qualified by its namespace prefix.
- 1424 – `qualifier-array-type` is typically the `qualifierSArray` type defined in the `common.xsd` file
- 1425 (but may be a user-defined type that complies with the mapping rules).

1426 EXAMPLE 1: A generalized example of the mapping of a multi-valued qualifier is as follows:

```
1427 <ns:QualifierName cim:qualifier="true">
1428   value
1429 </ns:QualifierName>
1430 ... // repeat QualifierName elements for each member of the array
```

1431 Where:

- 1432 – `QualifierName` is the name of the qualifier.
- 1433 – `ns` is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1434 – `n` is a sequential integer that represents the position of the entry in the array.
- 1435 – `value` is the string representation of the qualifier value.

1436 EXAMPLE 2: For example, the mapping of a `ValueMap` qualifier containing three entries ("OK", "Error",
 1437 "Unknown") is as follows:

```
1438 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
1439 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
1440 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
```

1441 A complete mapping of all qualifiers is provided in `qualifiers.xsd` (Annex A.2).

1442 11.2 Mapping CIM Qualifiers to XSD Elements

1443 All qualifiers are mapped using the normative rules in 11.1. The qualifiers listed in Table 8 are also
 1444 mapped directly into XSD features.

1445

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)]	<p>Mapped to a restriction using xs:maxLength on a string datatype. Required, with the following exception:</p> <p>A qualifier value of NULL shall not be mapped.</p> <p>For example:</p> <pre data-bbox="716 779 1421 1075"> <xs:element name="PropName"> <xs:complexType> <xs:simpleContent> <xs:restriction base="cim:cimString"> <xs:maxLength value="1024"/> <xs:anyAttribute .../> </xs:restriction> </xs:simpleContent> </xs:complexType> </xs:element> </pre>
MaxValue	[MaxValue (100)]	<p>Mapped to a restriction using xs:maxInclusive 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 1272 1421 1554"> <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>

CIM Qualifier	MOF Example	Mapped to XSD Structure
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="716 407 1414 695"><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="716 896 1414 1184"><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</p> <p><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>
Override	[Override ("PropName")]	<p>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).</p> <p>The following rules govern specific behavior regarding the inheritance of qualifiers (normatively defined in 11.3):</p> <ul data-bbox="764 1570 1409 1696" 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]	<p><code>nillable="false"</code> (normatively defined in 9.2.1.1)</p> <p>NOTE: If the effective value of the <code>Required</code> qualifier is false, then <code>nillable="true"</code> (required).</p>
ValueMap	[ValueMap (...)]	<p>ValueMaps may be mapped to XSD as an enumeration (see 9.2.3).</p>

1446 **11.3 Inheritance of Qualifiers**

1447 In addition to inheritance of properties, references, and methods through class inheritance, qualifier
 1448 values on any CIM elements are inherited. However, qualifiers are subject to special rules of inheritance.
 1449 Qualifier inheritance behavior is defined by the Flavors associated with a particular qualifier declaration.

1450 The rules covering qualifier inheritance are summarized in the third column of Table 9. In Table 9, the
 1451 term "overriding CIM elements in any subclasses" refers to CIM properties, references, and methods that
 1452 override other occurrences of the properties, references, or methods in their superclasses and therefore
 1453 form an inheritance chain. Note that duplicate property, reference, or method names that are *not*
 1454 overridden interrupt the inheritance chain for these CIM elements to their superclasses.

1455 **Table 9 – Rules of Qualifier Inheritance**

FLAVOR	Qualifier Inheritance Behavior (Informative)	Metadata Fragment Mapping Behavior
Restricted	<p>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.</p> <p>EXAMPLE: <code>Abstract</code></p>	<p>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.</p>
ToSubclass: EnableOverride	<p>The qualifier value is inherited by any subclasses or overriding CIM elements in these subclasses.</p> <p>The value of the qualifier may be changed in a subclass.</p> <p>EXAMPLE: <code>MaxLen</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> <p>In addition, the metadata fragment shall reflect the qualifier value provided on the corresponding CIM element. Unless overridden on the corresponding CIM element, the metadata fragment shall have the same value as the defined value in the superclass.</p>
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

1460 This annex provides examples of the WS-CIM Schema ([DSP8004](#)), the Qualifiers Schema ([DSP8005](#)),
1461 and the Class Hierarchy Type Schema ([DSP8006](#)).

1462 A.1 Common WS-CIM Schema: DSP8004

1463 This schema contains common definitions.

```

1464 <?xml version="1.0" encoding="utf-8" ?>
1465 <xs:schema targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1466   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1467   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1468   elementFormDefault="qualified">
1469
1470 <!-- The following are runtime attribute definitions -->
1471   <xs:attribute name="Key" type="xs:boolean"/>
1472
1473   <xs:attribute name="Version" type="xs:string"/>
1474
1475 <!-- The following section defines the extended WS-CIM datatypes -->
1476
1477   <xs:complexType name="cimDateTime">
1478     <xs:choice>
1479       <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
1480       <xs:element name="Interval" type="xs:duration"/>
1481       <xs:element name="Date" type="xs:date" />
1482       <xs:element name="Time" type="xs:time" />
1483       <xs:element name="Datetime" type="xs:dateTime"/>
1484     </xs:choice>
1485     <xs:anyAttribute namespace="##any" processContents="lax"/>
1486   </xs:complexType>
1487
1488   <xs:complexType name="cimUnsignedByte">
1489     <xs:simpleContent>
1490       <xs:extension base="xs:unsignedByte">
1491         <xs:anyAttribute namespace="##any" processContents="lax"/>
1492       </xs:extension>
1493     </xs:simpleContent>
1494   </xs:complexType>
1495
1496   <xs:complexType name="cimByte">
1497     <xs:simpleContent>
1498       <xs:extension base="xs:byte">
1499         <xs:anyAttribute namespace="##any" processContents="lax"/>
1500       </xs:extension>
1501     </xs:simpleContent>
1502   </xs:complexType>
1503
1504   <xs:complexType name="cimUnsignedShort">
1505     <xs:simpleContent>
1506       <xs:extension base="xs:unsignedShort">

```

```
1507     <xs:anyAttribute namespace="##any" processContents="lax"/>
1508   </xs:extension>
1509 </xs:simpleContent>
1510 </xs:complexType>
1511
1512 <xs:complexType name="cimShort">
1513   <xs:simpleContent>
1514     <xs:extension base="xs:short">
1515       <xs:anyAttribute namespace="##any" processContents="lax"/>
1516     </xs:extension>
1517   </xs:simpleContent>
1518 </xs:complexType>
1519
1520 <xs:complexType name="cimUnsignedInt">
1521   <xs:simpleContent>
1522     <xs:extension base="xs:unsignedInt">
1523       <xs:anyAttribute namespace="##any" processContents="lax"/>
1524     </xs:extension>
1525   </xs:simpleContent>
1526 </xs:complexType>
1527
1528 <xs:complexType name="cimInt">
1529   <xs:simpleContent>
1530     <xs:extension base="xs:int">
1531       <xs:anyAttribute namespace="##any" processContents="lax"/>
1532     </xs:extension>
1533   </xs:simpleContent>
1534 </xs:complexType>
1535
1536 <xs:complexType name="cimUnsignedLong">
1537   <xs:simpleContent>
1538     <xs:extension base="xs:unsignedLong">
1539       <xs:anyAttribute namespace="##any" processContents="lax"/>
1540     </xs:extension>
1541   </xs:simpleContent>
1542 </xs:complexType>
1543
1544 <xs:complexType name="cimLong">
1545   <xs:simpleContent>
1546     <xs:extension base="xs:long">
1547       <xs:anyAttribute namespace="##any" processContents="lax"/>
1548     </xs:extension>
1549   </xs:simpleContent>
1550 </xs:complexType>
1551
1552 <xs:complexType name="cimString">
1553   <xs:simpleContent>
1554     <xs:extension base="xs:string">
1555       <xs:anyAttribute namespace="##any" processContents="lax"/>
1556     </xs:extension>
1557   </xs:simpleContent>
1558 </xs:complexType>
1559
1560 <xs:complexType name="cimBoolean">
1561   <xs:simpleContent>
1562     <xs:extension base="xs:boolean">
1563       <xs:anyAttribute namespace="##any" processContents="lax"/>
1564     </xs:extension>
```

```

1565     </xs:simpleContent>
1566 </xs:complexType>
1567
1568 <xs:complexType name="cimFloat">
1569   <xs:simpleContent>
1570     <xs:extension base="xs:float">
1571       <xs:anyAttribute namespace="##any" processContents="lax"/>
1572     </xs:extension>
1573   </xs:simpleContent>
1574 </xs:complexType>
1575
1576 <xs:complexType name="cimDouble">
1577   <xs:simpleContent>
1578     <xs:extension base="xs:double">
1579       <xs:anyAttribute namespace="##any" processContents="lax"/>
1580     </xs:extension>
1581   </xs:simpleContent>
1582 </xs:complexType>
1583
1584 <xs:complexType name="cimChar16">
1585   <xs:simpleContent>
1586     <xs:restriction base="cim:cimString">
1587       <xs:maxLength value="1"/>
1588       <xs:anyAttribute namespace="##any" processContents="lax"/>
1589     </xs:restriction>
1590   </xs:simpleContent>
1591 </xs:complexType>
1592
1593 <xs:complexType name="cimBase64Binary">
1594   <xs:simpleContent>
1595     <xs:extension base="xs:base64Binary">
1596       <xs:anyAttribute namespace="##any" processContents="lax"/>
1597     </xs:extension>
1598   </xs:simpleContent>
1599 </xs:complexType>
1600
1601 <xs:complexType name="cimHexBinary">
1602   <xs:simpleContent>
1603     <xs:extension base="xs:hexBinary">
1604       <xs:anyAttribute namespace="##any" processContents="lax"/>
1605     </xs:extension>
1606   </xs:simpleContent>
1607 </xs:complexType>
1608
1609 <xs:complexType name="cimAnySimpleType">
1610   <xs:simpleContent>
1611     <xs:extension base="xs:anySimpleType">
1612       <xs:anyAttribute namespace="##any" processContents="lax"/>
1613     </xs:extension>
1614   </xs:simpleContent>
1615 </xs:complexType>
1616
1617 <xs:complexType name="cimReference">
1618   <xs:sequence>
1619     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
1620   </xs:sequence>
1621   <xs:anyAttribute namespace="##any" processContents="lax"/>
1622 </xs:complexType>

```

```

1623
1624 <!-- The following datatypes are used exclusively to define metadata fragments -->
1625 <xs:attribute name="qualifier" type="xs:boolean"/>
1626
1627 <xs:complexType name="qualifierString">
1628 <xs:simpleContent>
1629 <xs:extension base="cim:cimString">
1630 <xs:attribute ref="cim:qualifier" use="required"/>
1631 </xs:extension>
1632 </xs:simpleContent>
1633 </xs:complexType>
1634
1635 <xs:complexType name="qualifierBoolean">
1636 <xs:simpleContent>
1637 <xs:extension base="cim:cimBoolean">
1638 <xs:attribute ref="cim:qualifier" use="required"/>
1639 </xs:extension>
1640 </xs:simpleContent>
1641 </xs:complexType>
1642
1643 <xs:complexType name="qualifierUInt32">
1644 <xs:simpleContent>
1645 <xs:extension base="cim:cimUnsignedInt">
1646 <xs:attribute ref="cim:qualifier" use="required"/>
1647 </xs:extension>
1648 </xs:simpleContent>
1649 </xs:complexType>
1650
1651 <xs:complexType name="qualifierSInt64">
1652 <xs:simpleContent>
1653 <xs:extension base="cim:cimLong">
1654 <xs:attribute ref="cim:qualifier" use="required"/>
1655 </xs:extension>
1656 </xs:simpleContent>
1657 </xs:complexType>
1658
1659 <xs:complexType name="qualifierSArray">
1660 <xs:complexContent>
1661 <xs:extension base="cim:qualifierString"/>
1662 </xs:complexContent>
1663 </xs:complexType>
1664
1665 <!-- The following element is to be used only for defining metadata -->
1666 <xs:element name="DefaultValue" type="xs:anySimpleType" />
1667
1668 </xs:schema>

```

1669 A.2 Qualifiers Schema: DSP8005

1670 The following schema is an example of the qualifiers schema that is based on CIM Schema 2.13.1.
 1671 Future versions of CIM Schema may add or delete qualifiers, which would be reflected in the
 1672 corresponding qualifiers.xsd file.

```

1673 <?xml version="1.0" encoding="utf-8" ?>
1674 <xs:schema
1675 targetNamespace="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1676 xmlns:cimQ="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1677 xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"

```

```

1678     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1679     elementFormDefault="qualified">
1680
1681     <xs:import
1682         namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1683         schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1684
1685     <xs:element name="Abstract" type="cim:qualifierBoolean"/>
1686     <xs:element name="Aggregate" type="cim:qualifierBoolean"/>
1687     <xs:element name="Aggregation" type="cim:qualifierBoolean"/>
1688     <xs:element name="ArrayType" type="cim:qualifierString"/>
1689     <xs:element name="Association" type="cim:qualifierBoolean"/>
1690     <xs:element name="BitMap" type="cim:qualifierSArray"/>
1691     <xs:element name="BitValues" type="cim:qualifierSArray"/>
1692     <xs:element name="ClassConstraint" type="cim:qualifierSArray"/>
1693     <xs:element name="Counter" type="cim:qualifierBoolean"/>
1694     <xs:element name="Composition" type="cim:qualifierBoolean"/>
1695     <xs:element name="Deprecated" type="cim:qualifierSArray"/>
1696     <xs:element name="Description" type="cim:qualifierString"/>
1697     <xs:element name="DisplayName" type="cim:qualifierString"/>
1698     <xs:element name="DN" type="cim:qualifierBoolean"/>
1699     <xs:element name="EmbeddedInstance" type="cim:qualifierBoolean"/>
1700     <xs:element name="EmbeddedObject" type="cim:qualifierBoolean"/>
1701     <xs:element name="Exception" type="cim:qualifierBoolean"/>
1702     <xs:element name="Experimental" type="cim:qualifierBoolean"/>
1703     <xs:element name="Gauge" type="cim:qualifierBoolean"/>
1704     <xs:element name="In" type="cim:qualifierBoolean"/>
1705     <xs:element name="Indication" type="cim:qualifierBoolean"/>
1706     <xs:element name="Key" type="cim:qualifierBoolean"/>
1707     <xs:element name="MappingStrings" type="cim:qualifierSArray"/>
1708     <xs:element name="Max" type="cim:qualifierUInt32"/>
1709     <xs:element name="MethodConstraint" type="cim:qualifierSArray"/>
1710     <xs:element name="Min" type="cim:qualifierUInt32"/>
1711     <xs:element name="MaxLen" type="cim:qualifierUInt32"/>
1712     <xs:element name="MaxValue" type="cim:qualifierSInt64"/>
1713     <xs:element name="MinLen" type="cim:qualifierUInt32"/>
1714     <xs:element name="MinValue" type="cim:qualifierSInt64"/>
1715     <xs:element name="Revision" type="cim:qualifierString"/>           <!-- Is Deprecated -->
1716     <xs:element name="ModelCorrespondence" type="cim:qualifierSArray"/>
1717     <xs:element name="NullValue" type="cim:qualifierString"/>
1718     <xs:element name="OctetString" type="cim:qualifierBoolean"/>
1719     <xs:element name="Out" type="cim:qualifierBoolean"/>
1720     <xs:element name="Override" type="cim:qualifierString"/>
1721     <xs:element name="Propagated" type="cim:qualifierString"/>
1722     <xs:element name="PropertyConstraint" type="cim:qualifierSArray"/>
1723     <xs:element name="Read" type="cim:qualifierBoolean"/>
1724     <xs:element name="Required" type="cim:qualifierBoolean"/>
1725     <xs:element name="Schema" type="cim:qualifierString"/>
1726     <xs:element name="Static" type="cim:qualifierBoolean"/>
1727     <xs:element name="Terminal" type="cim:qualifierBoolean"/>
1728     <xs:element name="Units" type="cim:qualifierString"/>
1729     <xs:element name="UMLPackagePath" type="cim:qualifierString"/>
1730     <xs:element name="ValueMap" type="cim:qualifierSArray"/>

```

```

1731 <xs:element name="Values" type="cim:qualifierSArray"/>
1732 <xs:element name="Version" type="cim:qualifierString"/>
1733 <xs:element name="Weak" type="cim:qualifierBoolean"/>
1734 <xs:element name="Write" type="cim:qualifierBoolean"/>
1735
1736 <!-- Qualifier defined by DMTF for a future release of CIM Schema -->
1737 <!-- Included in this version at the request of the WSDM-CIM mapping team -->
1738 <xs:element name="Correlatable" type="cim:qualifierSArray"/>
1739
1740 <!-- Following qualifiers are considered to be "Optional Qualifiers" in CIM. -->
1741 <xs:element name="Alias" type="cim:qualifierString"/>
1742 <xs:element name="Delete" type="cim:qualifierBoolean"/>
1743 <xs:element name="Expensive" type="cim:qualifierBoolean"/>
1744 <xs:element name="IfDeleted" type="cim:qualifierBoolean"/>
1745 <xs:element name="Invisible" type="cim:qualifierBoolean"/>
1746 <xs:element name="Large" type="cim:qualifierBoolean"/>
1747 <xs:element name="Provider" type="cim:qualifierString"/>
1748 <xs:element name="PropertyUsage" type="cim:qualifierString"/>
1749 <xs:element name="Syntax" type="cim:qualifierString"/>
1750 <xs:element name="SyntaxType" type="cim:qualifierString"/>
1751 <xs:element name="TriggerType" type="cim:qualifierString"/>
1752 <xs:element name="UnknownValues" type="cim:qualifierSArray"/>
1753 <xs:element name="UnsupportedValues" type="cim:qualifierSArray"/>
1754
1755 </xs:schema>

```

1756 A.3 Class Hierarchy Type Schema: DSP8006

1757 The complex type definition in the following schema provides the type of GEDs that describe the CIM
 1758 Schema subclass / superclass hierarchy. The element `ClassHierarchy` may be used by protocols as
 1759 an element in XML instance documents of a CIM instance that contains a value representing the subclass
 1760 / superclass hierarchy of a class. Its presence as an element in an instance document would be covered
 1761 by the `xs:any` in the WS-CIM schema of the instance's class.

```

1762 <?xml version="1.0" encoding="utf-8"?>
1763 <xs:schema
1764   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1765   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1766   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1767   <xs:complexType name="ClassHierarchyType">
1768     <xs:sequence>
1769       <xs:any minOccurs="0" namespace="##any" processContents="lax" />
1770     </xs:sequence>
1771   </xs:complexType>
1772
1773   <xs:element name="ClassHierarchy" type="ctype:ClassHierarchyType"/>
1774
1775 </xs:schema>

```


ANNEX B (Informative)

Examples

1780 This annex contains examples of converting MOF definitions of several classes into XML Schema, WSDL
1781 fragments, and metadata fragments. Although the classes are fictional creations used to illustrate
1782 different features of the conversion, the classes are based on actual CIM classes.

1783 B.1 MOF Definitions

1784 This clause contains the MOF definitions that are converted in these examples.

1785 B.1.1 EX_BaseComponent

```
1786 [Abstract, Version ( "2.x" ), Description (
1787     "EX_BaseComponent serves as an example base CIM class.")]
1788 class EX_BaseComponent {
1789     [Description (
1790         "A datetime value indicating when the object was installed.")]
1791     datetime InstallDate;
1792     [Description (
1793         "The Name property defines the label by which the object is "
1794         "known."),
1795         MaxLen ( 1024 ), Required]
1796     string Name;
1797     [Description (
1798         "A set of descriptive statements that can be used to describe the "
1799         "state of a Component."),
1800         ArrayType ( "Indexed" ) ]
1801     string StatusDescriptions[];
1802     [Description (
1803         "A descriptive code representing operational health of a Component."),
1804         ValueMap { "OK", "Error", "Unknown" }, MaxLen ( 10 ) ]
1805     string HealthStatus;
1806 };
```

1807 B.1.2 EX_DerivedComponent

```
1808 [Version ( "2.x" ), Description (
1809     "This class extends EX_BaseComponent.")]
1810 class EX_DerivedComponent : EX_BaseComponent {
1811     [Description (
1812         "EnabledState is an integer enumeration that indicates the "
1813         "enabled and disabled states of a derived Component."),
1814         ValueMap { "0", "1", "2", "3" },
1815         Values { "Unknown", "Other", "Enabled", "Disabled" } ]
1816     uint16 EnabledState=3;
1817     [Description (
1818         "Boolean flag indicating availability of a Component.") ]
1819     boolean AvailableFlag;
1820     [Description (
1821         "Requests that the state of the element be changed to the "
1822         "value specified in the RequestedState parameter."),
1823         ValueMap { "0", "1", "2", "3..32767", "32768..65535" },
1824         Values { "Completed with No Error", "Not Supported",
```

```

1825         "Failed", "DMTF Reserved", "Vendor Specific" } ]
1826     uint32 RequestStateChange(
1827         [IN, Description (
1828             "The state requested for the Component."),
1829             ValueMap { "2", "3", "4" },
1830             Values { "Enabled", "Disabled" "Shutdown" } ]
1831     uint16 RequestedState,
1832     [IN ( false ), OUT, Description (
1833         "Reference to an instance of some class (undefined in this "
1834         "example) that is returned upon completion of the operation."))
1835     CIM_SomeClass REF ResultClass,
1836     [IN, Description (
1837         "A timeout period that specifies the maximum amount of "
1838         "time that the client expects the transition to the new "
1839         "state to take. ")]
1840     datetime TimeoutPeriod);
1841 };

```

1842 B.1.3 EX_AssociationComponent

```

1843     [Association, Version ( "2.x" ), Description (
1844         "Indicates that two entites are associated.")]
1845 class EX_Association {
1846     [Key, Description (
1847         "AssociatingComponent represents one Component is "
1848         "associated with the Component referenced as AssociatedComponent.")]
1849     EX_BaseComponent REF AssociatingComponent;
1850     [Key, Description (
1851         "AssociatedComponent represents another Component (up to 4) that "
1852         "is associated with the Component referenced as "
1853         "AssociatingComponent."),
1854         Max ( 4 )]
1855     EX_BaseComponent REF AssociatedComponent;
1856     [Description (
1857         "The point in time that the Components were associated.")]
1858     datetime WhenAssociated;
1859     [Description (
1860         "Boolean indicating whether the association is maintained.")]
1861     boolean AssocMaintained;
1862 };

```

1863 B.2 XSD

1864 This clause shows the XML Schema files that would result from the application of this specification to the
 1865 preceding example CIM classes.

1866 B.2.1 EX_BaseComponent

```

1867 <?xml version="1.0" encoding="utf-8"?>
1868 <xs:schema
1869     targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1870     xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1871     xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1872     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1873     ...>
1874 <xs:import
1875     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1876     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1877 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>

```

```

1878 <xs:element name="Name">
1879   <xs:complexType>
1880     <xs:simpleContent>
1881       <xs:restriction base="cim:cimString">
1882         <xs:maxLength value="1024"/>
1883         <xs:anyAttribute namespace="##any" processContents="lax"/>
1884       </xs:restriction>
1885     </xs:simpleContent>
1886   </xs:complexType>
1887 </xs:element>
1888 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1889 <xs:element name="HealthStatus" nillable="true">
1890   <xs:complexType>
1891     <xs:simpleContent>
1892       <xs:restriction base="cim:cimString">.
1893         <xs:enumeration value="OK"/>
1894         <xs:enumeration value="Error"/>
1895         <xs:enumeration value="Unknown"/>
1896         <xs:maxLength value="10"/>
1897         <xs:anyAttribute namespace="##any" processContents="lax"/>
1898       </xs:restriction>
1899     </xs:simpleContent>
1900   </xs:complexType>
1901 </xs:element>
1902 <xs:complexType name="EX_BaseComponent_Type">
1903   <xs:sequence>
1904     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1905     <xs:element ref="class:InstallDate" minOccurs="0"/>
1906     <xs:element ref="class:Name"/>
1907     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1908     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1909   </xs:sequence>
1910   <xs:anyAttribute namespace="##any" processContent="lax"/>
1911 </xs:complexType>
1912 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type"/>
1913 </xs:schema>

```

1914 B.2.2 EX_DerivedComponent

```

1915 <?xml version="1.0" encoding="utf-8"?>
1916 <xs:schema
1917   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1918   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1919   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1920   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1921   ...>
1922 <xs:import
1923   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1924   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1925 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
1926 <xs:element name="Name">
1927   <xs:complexType>
1928     <xs:simpleContent>
1929       <xs:restriction base="cim:cimString">
1930         <xs:maxLength value="1024"/>
1931         <xs:anyAttribute namespace="##any" processContents="lax"/>
1932       </xs:restriction>
1933     </xs:simpleContent>
1934   <xs:complexType>

```

```

1935 </xs:element>
1936 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1937 <xs:element name="HealthStatus" nillable="true">
1938   <xs:complexType>
1939     <xs:simpleContent>
1940       <xs:restriction base="cim:cimString">
1941         <xs:enumeration value="OK"/>
1942         <xs:enumeration value="Error"/>
1943         <xs:enumeration value="Unknown"/>
1944         <xs:maxLength value="10"/>
1945         <xs:anyAttribute namespace="##any" processContents="lax"/>
1946       </xs:restriction>
1947     </xs:simpleContent>
1948   </xs:complexType>
1949 </xs:element>
1950 <xs:element name="EnabledState" nillable="true">
1951   <xs:complexType>
1952     <xs:simpleContent>
1953       <xs:restriction base="cim:cimUnsignedShort">
1954         <xs:enumeration value="0"/>
1955         <xs:enumeration value="1"/>
1956         <xs:enumeration value="2"/>
1957         <xs:enumeration value="3"/>
1958         <xs:anyAttribute namespace="##any" processContents="lax"/>
1959       </xs:restriction>
1960     </xs:simpleContent>
1961   </xs:complexType>
1962 </xs:element>
1963 <xs:element name="AvailableFlag" type="cim:cimBoolean" nillable="true"/>
1964 <xs:complexType name="EX_DerivedComponent_Type">
1965   <xs:sequence>
1966     <xs:element ref="class:AvailableFlag" minOccurs="0"/>
1967     <xs:element ref="class:EnabledState" minOccurs="0"/>
1968     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1969     <xs:element ref="class:InstallDate" minOccurs="0"/>
1970     <xs:element ref="class:Name"/>
1971     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1972     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1973   </xs:sequence>
1974   <xs:anyAttribute namespace="##any" processContent="lax"/>
1975 </xs:complexType>
1976 <xs:element name="EX_DerivedComponent" type="class:EX_DerivedComponent_Type"/>
1977 <xs:element name="RequestStateChange_INPUT">
1978   <xs:complexType>
1979     <xs:sequence>
1980       <xs:element name="RequestedState" nillable="true">
1981         <xs:complexType>
1982           <xs:simpleContent>
1983             <xs:restriction base="cim:cimUnsignedShort">
1984               <xs:enumeration value="2"/>
1985               <xs:enumeration value="3"/>
1986               <xs:enumeration value="4">
1987                 <xs:anyAttribute namespace="##any" processContents="lax"/>
1988               </xs:restriction>
1989             </xs:simpleContent>
1990           </xs:complexType>
1991         </xs:element>
1992       <xs:element name="TimeoutPeriod" type="cim:cimDateTime" nillable="true"/>

```

```

1993     </xs:sequence>
1994   </xs:complexType>
1995 </xs:element>
1996 <xs:element name="RequestStateChange_OUTPUT">
1997   <xs:complexType>
1998     <xs:sequence>
1999       <xs:element name="ResultClass" type="cim:cimReference" nillable="true"/>
2000       <xs:element name="ReturnValue" nillable="true">
2001         <xs:complexType>
2002           <xs:simpleContent>
2003             <xs:restriction base="cim:cimAnySimpleType">
2004               <xs:simpleType>
2005                 <xs:union>
2006                   <xs:simpleType>
2007                     <xs:restriction base="xs:unsignedInt">
2008                       <xs:enumeration value="0"/>
2009                       <xs:enumeration value="1"/>
2010                       <xs:enumeration value="2"/>
2011                     </xs:restriction>
2012                   </xs:simpleType>
2013                   <xs:simpleType>
2014                     <xs:restriction base="xs:unsignedInt">
2015                       <xs:minInclusive value="3"/>
2016                       <xs:maxInclusive value="32767"/>
2017                     </xs:restriction>
2018                   </xs:simpleType>
2019                   <xs:simpleType>
2020                     <xs:restriction base="xs:unsignedInt">
2021                       <xs:minInclusive value="32768"/>
2022                       <xs:maxInclusive value="65535"/>
2023                     </xs:restriction>
2024                   </xs:simpleType>
2025                 </xs:union>
2026               </xs:simpleType>
2027               <xs:anyAttribute namespace="##any" processContents="lax"/>
2028             </xs:restriction>
2029           </xs:simpleContent>
2030         </xs:complexType>
2031       </xs:element>
2032     </xs:sequence>
2033   </xs:complexType>
2034 </xs:element>
2035 </xs:schema>

```

2036 **B.2.3 EX_AssociationComponent**

```

2037 <?xml version="1.0" encoding="utf-8"?>
2038 <xs:schema
2039   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2040   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
2041   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2042   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2043   ...>
2044 <xs:import
2045   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
2046   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
2047 <xs:element name="AssociatingComponent" type="cim:cimReference"/>
2048 <xs:element name="AssociatedComponent" type="cim:cimReference"/>
2049 <xs:element name="WhenAssociated" type="cim:cimDateTime" nillable="true"/>

```

```

2050 <xs:element name="AssocMaintained" type="cim:cimBoolean" nillable="true"/>
2051 <xs:complexType name="EX_AssociationComponent_Type">
2052   <xs:sequence>
2053     <xs:element ref="class:AssociatedComponent" />
2054     <xs:element ref="class:AssociatingComponent" />
2055     <xs:element ref="class:AssocMaintained" minOccurs="0"/>
2056     <xs:element ref="class:WhenAssociated" minOccurs="0"/>
2057     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2058   </xs:sequence>
2059   <xs:anyAttribute namespace="##any" processContent="lax"/>
2060 </xs:complexType>
2061 <xs:element name="EX_AssociationComponent" type="class:EX_AssociationComponent_Type"/>
2062 </xs:schema>

```

2063 B.2.4 Class Hierarchy Schema

```

2064 <?xml version="1.0" encoding="utf-8"?>
2065 <xs:schema
2066   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2067   xmlns:chier="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2068   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2069   xmlns:xs="http://www.w3.org/2001/XMLSchema">
2070 <xs:import
2071   namespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2072   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/classhiertype.xsd"/>
2073 <xs:element name="EX_BaseComponent_Class">
2074   <xs:complexType>
2075     <xs:complexContent>
2076       <xs:restriction base="ctype:ClassHierarchyType" />
2077     </xs:complexContent>
2078   </xs:complexType>
2079 </xs:element>
2080 <xs:element name="EX_DerivedComponent_Class">
2081   <xs:complexType>
2082     <xs:complexContent>
2083       <xs:restriction base="ctype:ClassHierarchyType">
2084         <xs:sequence>
2085           <xs:element ref="chier:EX_BaseComponent_Class" />
2086         </xs:sequence>
2087       </xs:restriction>
2088     </xs:complexContent>
2089   </xs:complexType>
2090 </xs:element>
2091 <xs:element name="EX_AssociationComponent_Class">
2092   <xs:complexType>
2093     <xs:complexContent>
2094       <xs:restriction base="ctype:ClassHierarchyType" />
2095     </xs:complexContent>
2096   </xs:complexType>
2097 </xs:element>
2098 </xs:schema>

```

2099 B.3 WSDL Fragments

2100 This clause contains the WSDL fragments (`wSDL:message`, `wSDL:operation`) that would result from
 2101 the application of this specification to the `EX_DerivedComponent` class. This class specifies only one
 2102 method, `RequestStateChange`.

```

2103 <?xml version="1.0" encoding="utf-8"?>
2104 <wSDL:definitions
2105     xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
2106     targetNamespace="http://. . .wSDL"
2107     xmlns:cimClass="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
2108     xmlns:thisWSDL="http://. . .wSDL"
2109     ...>
2110 <w:import namespace="http://www.w3.org/2005/08/addressing"
2111     location="http://www.w3.org/2005/08/addressing/ws-addr.xsd"/>
2112 <wSDL:types>
2113     ... <!-- Schema of EX_DerivedComponent -->
2114 </wSDL:types>
2115 <wSDL:message name="RequestStateChange_InputMessage">
2116     <wSDL:part name="body"
2117         element="cimClass:RequestStateChange_INPUT"/>
2118 </wSDL:message>
2119 <wSDL:message name="RequestStateChange_OutputMessage">
2120     <wSDL:part name="body"
2121         element="cimClass:RequestStateChange_OUTPUT"/>
2122 </wSDL:message>
2123 <!-- OPERATION: RequestStateChange
2124     <wSDL:operation name="RequestStateChange">
2125         <wSDL:input name="RequestStateChange_InputMessage"
2126             message="thisWSDL:RequestStateChange_InputMessage"
2127         <wSDL:output name="RequestStateChange_OutputMessage"
2128             message="thisWSDL:RequestStateChange_OutputMessage"
2129         </wSDL:operation>
2130 -->
2131 </wSDL:definitions>
  
```

2132 B.4 MetaData Fragments

2133 Metadata fragments are generated from the qualifiers that are associated with a class, property,
 2134 reference, method, or parameter. XML documents that incorporate these fragments must import the `cim`
 2135 and `cimQ` namespaces.

2136 B.4.1 EX_BaseComponent

2137 B.4.1.1 Class Qualifiers

```

2138 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
2139 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2140 <cimQ:Description cim:qualifier="true">
2141     EX_BaseComponent serves as an example base CIM class.
2142 </cimQ:Description>
  
```

2143 B.4.1.2 Property Qualifiers

2144 B.4.1.2.1 HealthStatus

```

2145 <cimQ:Description cim:qualifier="true">
2146     A descriptive code of the operational health of a Component.
2147 </cimQ:Description>
2148 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
  
```

2149 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>

2150 **B.4.1.2.2 InstallDate**

2151 <cimQ:Description cim:qualifier="true">
 2152 EX_BaseComponent serves as an example base CIM class.
 2153 </cimQ:Description>
 2154 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>

2155 **B.4.1.2.3 Name**

2156 <cimQ:Description cim:qualifier="true">
 2157 The Name property defines the label by which the object is known.
 2158 </cimQ:Description>
 2159 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
 2160 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

2161 **B.4.1.2.4 StatusDescriptions**

2162 <cimQ:Description cim:qualifier="true">
 2163 A set of descriptive statements that can be used to describe the state of an Component.
 2164 </cimQ:Description>
 2165 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

2166 **B.4.2 EX_DerivedComponent**

2167 **B.4.2.1 Class Qualifiers**

2168 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
 2169 <cimQ:Description cim:qualifier="true">
 2170 This class extends EX_BaseComponent.
 2171 </cimQ:Description>

2172 **B.4.2.2 Property Qualifiers**

2173 **B.4.2.2.1 AvailableFlag**

2174 <cimQ:Description cim:qualifier="true">
 2175 Boolean flag indicating availability of a Component.

2176 **B.4.2.2.2 EnabledState**

2177 <cimQ:Description cim:qualifier="true">
 2178 EnabledState is an integer enumeration that indicates the enabled
 2179 and disabled states of a derived Component.
 2180 </cimQ:Description>
 2181 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
 2182 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
 2183 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
 2184 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
 2185 <cimQ:Values cim:qualifier="true">Unknown</cimQ:Values>
 2186 <cimQ:Values cim:qualifier="true">Other</cimQ:Values>
 2187 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
 2188 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
 2189 <cim:DefaultValue xsi:type="xs:uint16">3</cim:DefaultValue>
 2190 HealthStatus
 2191 <cimQ:Description cim:qualifier="true">
 2192 A descriptive code of the operational health of a Component.
 2193 </cimQ:Description>
 2194 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
 2195 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>


```

2196 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
2197 </cimQ:Description>

```

2198 **B.4.2.2.3 InstallDate**

```

2199 <cimQ:Description cim:qualifier="true">
2200     EX_BaseComponent serves as an example base CIM class.
2201 </cimQ:Description>

```

2202 **B.4.2.2.4 Name**

```

2203 <cimQ:Description cim:qualifier="true">
2204     The Name property defines the label by which the object is known.
2205 </cimQ:Description>
2206 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
2207 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

```

2208 **B.4.2.2.5 StatusDescriptions**

```

2209 <cimQ:Description cim:qualifier="true">
2210     A set of descriptive statements that can used to describe the state of an Component.
2211 </cimQ:Description>
2212 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

```

2213 **B.4.2.2.6 AvailableFlag**

```

2214 <cimQ:Description cim:qualifier="true">
2215     Boolean flag indicating availability of a Component.
2216 </cimQ:Description>

```

2217 **B.4.2.3 Method and Parameter Qualifiers**

2218 **B.4.2.3.1 RequestStatusChange Method**

```

2219 <cimQ:Description cim:qualifier="true">
2220     Requests that the state of the element be changed to the value
2221     specified in the RequestedState parameter.
2222 </cimQ:Description>
2223 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
2224 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
2225 <cimQ:ValueMap cim:qualifier="true">..</cimQ:ValueMap>
2226 <cimQ:ValueMap cim:qualifier="true">4096</cimQ:ValueMap>
2227 <cimQ:ValueMap cim:qualifier="true">4100..32767</cimQ:ValueMap>
2228 <cimQ:ValueMap cim:qualifier="true">32768..65535</cimQ:ValueMap>
2229 <cimQ:Values cim:qualifier="true">Completed with No Error</cimQ:Values>
2230 <cimQ:Values cim:qualifier="true">Not Supported</cimQ:Values>
2231 <cimQ:Values cim:qualifier="true">Unknown or Unspecified Error</cimQ:Values>
2232 <cimQ:Values cim:qualifier="true">Failed</cimQ:Values>
2233 <cimQ:Values cim:qualifier="true">DMTF Reserved</cimQ:Values>
2234 <cimQ:Values cim:qualifier="true">Vendor Specific</cimQ:Values>

```

2235 **B.4.2.3.2 RequestedState Parameter**

```

2236 <cimQ:Description cim:qualifier="true">
2237     The state requested for the Component.
2238 </cimQ:Description>
2239 <cimQ:In cim:qualifier="true">true</cimQ:In>
2240 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
2241 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
2242 <cimQ:ValueMap cim:qualifier="true">4</cimQ:ValueMap>
2243 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>

```

2244 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>

2245 <cimQ:Values cim:qualifier="true">Shutdown</cimQ:Values>

2246 **B.4.2.3.3 ResultClass Parameter**

2247 <cimQ:Description cim:qualifier="true">

2248 Reference to an instance of some class (undefined in this example)
2249 that is returned upon completion of the operation.

2250 </cimQ:Description>

2251 <cimQ:Out cim:qualifier="true">true</cimQ:Out>

2252 <cimQ:In cim:qualifier="true">false</cimQ:In>

2253 **B.4.2.3.4 TimeoutPeriod Parameter**

2254 <cimQ:Description cim:qualifier="true">

2255 A timeout period that specifies the maximum amount of time that the
2256 client expects the transition to the new state to take.

2257 </cimQ:Description>

2258 <cimQ:In cim:qualifier="true">true</cimQ:In>

2259 **B.4.3 EX_AssociationComponent**

2260 **B.4.3.1 Class Qualifiers**

2261 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>

2262 <cimQ:Description cim:qualifier="true">

2263 Indicates that two entites are associated.

2264 </cimQ:Description>

2265 <cimQ:Association cim:qualifier="true">true</cimQ:Association>

2266 **B.4.3.2 Property Qualifiers**

2267 **B.4.3.2.1 AssociatedComponent**

2268 <cimQ:Key cim:qualifier="true">true</cimQ:Key>

2269 <cimQ:Description cim:qualifier="true">

2270 AssociatedComponent represents another Component (up to 4) that is associated
2271 with the Component referenced as AssociatingComponent.

2272 </cimQ:Description>

2273 <cimQ:Max cim:qualifier="true">4</cimQ:Max>

2274 **B.4.3.2.2 AssociatingComponent**

2275 <cimQ:Key cim:qualifier="true">true</cimQ:Key>

2276 <cimQ:Description cim:qualifier="true">

2277 An AssociatingComponent represents one Component is associated with the
2278 component referenced as AssociatedComponent.

2279 </cimQ:Description>

2280 **B.4.3.2.3 AssocMaintained**

2281 <cimQ:Description cim:qualifier="true">

2282 Boolean indicating whether the association is maintained.

2283 </cimQ:Description>

2284 **B.4.3.2.4 WhenAssociated**

2285 <cimQ:Description cim:qualifier="true">

2286 The point in time that the Components were associated.

2287 </cimQ:Description>

ANNEX C (Informative)

Collation Optimization Available to Implementers

2288
2289
2290
2291

2292 **C.1 Sorting with Limited Character Set**

2293 The character set permitted in CIM identifiers is limited to

- 2294 - U+0030..U+0039 (digits 0-9)
- 2295 - U+0041..U+005A (alphabetics A-Z)
- 2296 - U+0061..U+007A (alphabetics a-z)
- 2297 - U+0052 (underscore)
- 2298 - U+0080..U+FFEF (the rest of Unicode)

2299 The intention of the preferred collation is that the property name identifier strings should be ordered by a
2300 binary sorting of big-endian UCS-2 representation of the characters. For example, ABC, ABc, and AbC
2301 sort in this order because

2302 ABC 00 41 00 42 00 43

2303 ABc 00 41 00 42 00 63

2304 AbC 00 41 00 62 00 43

2305 (and accented As sort in order by Unicode number)

2306 If the identifiers of a MOF do not use the full range of characters, optimization can be applied to the
2307 sorting of property identifiers. Specifically, a simple approach exists for the commonly observed case
2308 where MOF characters are limited to ASCII-7, i.e., character values < 0x7F.

2309 Most or all MOFs that you will encounter contain only (a subset of) ASCII-7 characters, that is, characters
2310 in the range 0x00 to 0x7F. For such MOFs, it is not necessary to cast the property identifier strings into
2311 Unicode representation at all. If all the characters are ASCII-7, then the strings can be sorted as simple
2312 one-byte sequences.

2313 If the internal representation of character strings (in a CIMOM, protocol adapter, or client application, etc.)
2314 is UTF-8, note that ASCII-7 characters are represented unmodified in UTF-8.

2315 Suggested algorithm: Pre-scan the MOF, or at least the set of property identifiers, for characters > 0x7F.
2316 If the set contains no such characters > 0x7F, then sort the strings as simple one-byte octet strings.

2317 This case can be used for all currently published DMTF MOFs.

2318 **C.2 Note of Caution Concerning Collation**

2319 The default Unicode Collation Algorithm orders properties differently from the current DMTF practice. All
2320 published CIM XML schemata order properties as described here (sorting by Unicode value). The UCA,
2321 by default, uses a different ordering even for the subset of ASCII-7 characters: lower case characters sort
2322 before upper case characters. This results in differences in some cases, where, for example, "CPU" and
2323 "Caption" appear in one order in the published XSD files but sort in the other order using the UCA.

2324 The intention of the preferred algorithm is to retain current DMTF practice. Properties in XSDs will
2325 continue to be in upper-before-lower order. Some existing WS-Man service implementations may be
2326 using the default Unicode Collation Algorithm. These implementations will have to become compatible
2327 with existing practice.

**ANNEX D
(Informative)****Change Log**

Version	Date	Description
1.0.0	2008-05-22	Released as Final Standard
1.0.1	2009-04-21	Released as DMTF Standard, with the following changes: <ul style="list-style-type: none">• Clarified ambiguous references to WS-Addressing specifications in clause 3• Clarified rules for collation sequence of properties in subclause 9.3.1 and in the new ANNEX C
1.0.2	2010-02-08	Released as DMTF Standard, with the following changes: <ul style="list-style-type: none">• Changed the representation of octet strings in subclause 9.2.4 to remove the length representation

2328
2329
2330
2331
2332

2333

2334

Bibliography

2335 DMTF DSP4009, *Process for publishing XML schema, XML documents and XSLT stylesheets 1.0*,
2336 http://www.dmtf.org/standards/published_documents/DSP4009_1.0.pdf

2337