



1

2

3

4

**Document Number: DSP0230**

**Date: 2009-04-21**

**Version: 1.0.1**

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, 2009 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26 withdrawn or modified after publication, and shall be indemnified and held harmless by any party  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 | 3.1 Approved References .....  | 8  |
| 42 | 3.2 Other References.....  | 9  |
| 43 | 4 Terms and Definitions.....   | 9  |
| 44 | 5 Symbols and Abbreviated Terms.....   | 10 |
| 45 | 6 Namespace Prefixes and Schema Locations .....                              | 11 |
| 46 | 7 Dereferencing Schema URI Locations in Order to Access XML Schema.....      | 13 |
| 47 | 8 Mapping Primitive Datatypes .....  | 14 |
| 48 | 8.1 cimDateTime Datatype .....   | 15 |
| 49 | 8.2 CIM References .....   | 16 |
| 50 | 8.3 cimAnySimpleType Datatype.....   | 17 |
| 51 | 8.4 Derivation by Restriction .....  | 17 |
| 52 | 8.5 WS-CIM Canonical Values .....  | 17 |
| 53 | 9 CIM Class to XML Schema Mappings .....                                     | 18 |
| 54 | 9.1 Class Namespace.....   | 18 |
| 55 | 9.2 Properties.....  | 18 |
| 56 | 9.3 Class Structure .....  | 27 |
| 57 | 9.4 Class Inheritance .....  | 29 |
| 58 | 9.5 Method Parameters .....  | 30 |
| 59 | 9.6 CIM Instances .....  | 33 |
| 60 | 9.7 Superclass Class Representation.....                                     | 33 |
| 61 | 10 CIM Methods to WSDL Mappings.....   | 35 |
| 62 | 10.1 Defining WSDL Message Structures .....                                  | 35 |
| 63 | 10.2 Defining WSDL Operation Structures .....                                | 35 |
| 64 | 10.3 Defining wsa:Actions.....   | 36 |
| 65 | 11 Qualifier Mappings .....  | 37 |
| 66 | 11.1 General Format.....   | 37 |
| 67 | 11.2 Mapping CIM Qualifiers to XSD Elements.....                             | 39 |
| 68 | 11.3 Inheritance of Qualifiers .....   | 41 |
| 69 | ANNEX A (Informative) Schemas .....  | 43 |
| 70 | A.1 Common WS-CIM Schema: DSP8004 .....                                      | 43 |
| 71 | A.2 Qualifiers Schema: DSP8005 .....   | 46 |
| 72 | A.3 Class Hierarchy Type Schema: DSP8006.....                                | 48 |
| 73 | ANNEX B (Informative) Conventions .....                                      | 49 |
| 74 | ANNEX C (Informative) Examples .....   | 50 |
| 75 | C.1 MOF Definitions .....  | 50 |
| 76 | C.2 XSD.....   | 51 |
| 77 | C.3 WSDL Fragments .....   | 55 |
| 78 | C.4 MetaData Fragments .....   | 56 |
| 79 | ANNEX D (Informative) Collation Optimization Available to Implementers ..... | 60 |
| 80 | D.1 Sorting with Limited Character Set .....                                 | 60 |
| 81 | D.2 Note of Caution Concerning Collation .....                               | 60 |
| 82 |  |    |

83 **Tables**

84 Table 1 – Namespaces ..... 11

85 Table 2 – Namespace Prefixes ..... 12

86 Table 3 – Schema URI Locations ..... 12

87 Table 4 – XSD DSP Numbers..... 12

88 Table 5 – Mapping CIM Datatypes to WS-CIM Datatypes ..... 14

89 Table 6 – Rules for Converting datetime to cimDateTime ..... 15

90 Table 7 – Rules for Converting cimDateTime to datetime ..... 15

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

92 Table 9 – Rules of Qualifier Inheritance..... 42

93

94

## Foreword

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

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

99 Authors:

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

119

120

## Introduction

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

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

133

# WS-CIM Mapping Specification

## 134 **1 Scope**

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

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

### 144 **1.1 In-Scope Features**

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

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

### 152 **1.2 Out-of-Scope Considerations**

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

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

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

## 177 2 Conformance

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

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

## 182 3 Normative References

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

### 186 3.1 Approved References

187 DMTF DSP0004, *Common Information Model (CIM) Infrastructure Specification, 2.3*  
188 [http://www.dmtf.org/standards/published\\_documents/DSP0004V2.3\\_final.pdf](http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf)

189 DMTF DSP0226, *Web Services for Management Specification, 1.1.0*,  
190 [http://www.dmtf.org/standards/published\\_documents/DSP0226\\_1.1.0.pdf](http://www.dmtf.org/standards/published_documents/DSP0226_1.1.0.pdf)

191 DMTF DSP4009, *Process for publishing XML schema, XML documents and XSLT stylesheets*  
192 [http://www.dmtf.org/standards/published\\_documents/DSP4009\\_1.0.0.pdf](http://www.dmtf.org/standards/published_documents/DSP4009_1.0.0.pdf)

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

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

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

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

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

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

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

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



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

## 211 **3.2 Other References**

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

## 214 **4 Terms and Definitions**

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

### 216 **3.1**

#### 217 **can**

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

### 219 **3.2**

#### 220 **cannot**

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

### 222 **3.3**

#### 223 **conditional**

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

### 226 **3.4**

#### 227 **mandatory**

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

### 230 **3.5**

#### 231 **may**

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

### 233 **3.6**

#### 234 **need not**

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

### 236 **3.7**

#### 237 **optional**

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

### 239 **3.8**

#### 240 **shall**

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

### 243 **3.9**

#### 244 **shall not**

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

- 247 **3.10**  
248 **should**  
249 indicates that among several possibilities, one is recommended as particularly suitable, without  
250 mentioning or excluding others, or that a certain course of action is preferred but not necessarily required
- 251 **3.11**  
252 **should not**  
253 indicates that a certain possibility or course of action is deprecated but not prohibited
- 254 **3.12**  
255 **Global Element Declaration**  
256 element declaration in an XML Schema that places the element as an immediate child of the root element  
257 of the schema
- 258 **3.13**  
259 **Managed Object Format**  
260 an IDL based language, defined by the DMTF, expressing the structure, behavior, and semantics of a  
261 CIM class and its instances
- 262 **3.14**  
263 **Uniform Record Identifier**  
264 a Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or  
265 physical resource. See [RFC3986](#).
- 266 **3.15**  
267 **Runtime**  
268 describes the situation where XML instances are produced that are conformant to this specification
- 269 **3.16**  
270 **WS-CIM**  
271 of or pertaining to this specification  
272 NOTE: "WS-CIM" is not an acronym; it should be treated simply as the name of the contents of the specification.
- 273 **3.17**  
274 **XML instance document**  
275 an XML document that conforms to a specified XML Schema  
276 As used in this specification, *XML instance document* refers to a document that conforms to an XML  
277 Schema that conforms to the rules in clause 9.
- 278 **3.18**  
279 **XSDL**  
280 offers facilities for describing the structure and constraining the contents of XML documents, including  
281 those which exploit the XML Namespace facility. XSDL documents have the '.xsd' file extension. See:  
282 <http://www.w3.org/TR/xmlschema11-1/>.

## 283 **5 Symbols and Abbreviated Terms**

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

- 285 **5.1**  
286 **GED**  
287 Global Element Declaration

- 288 **5.2**
- 289 **MOF**
- 290 Managed Object Format
- 291 **5.3**
- 292 **URI**
- 293 Uniform Resource Identifier
- 294 **5.4**
- 295 **WSDL**
- 296 Web Services Description Language
- 297 **5.5**
- 298 **XML**
- 299 Extensible Mark-up Language
- 300 **5.6**
- 301 **XSDL**
- 302 XML Schema Definition Language

303 **6 Namespace Prefixes and Schema Locations**

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

306 This specification defines namespaces as shown in Table 1.

307 **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  |

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

310 **Table 2 – Namespace Prefixes**

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

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

312 **Table 3 – Schema URI Locations**

| Prefix | Schema Location URLs  |
|--------|---|
| class  | <a href="http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/ClassName.xsd">http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/ClassName.xsd</a>           |
| cim    | <a href="http://schemas.dmtf.org/wbem/wscim/X/common.xsd">http://schemas.dmtf.org/wbem/wscim/X/common.xsd</a>   |
| cimQ   | <a href="http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/qualifiers.xsd">http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/qualifiers.xsd</a>         |
| ctype  | <a href="http://schemas.dmtf.org/wbem/wscim/X/classhiertype.xsd">http://schemas.dmtf.org/wbem/wscim/X/classhiertype.xsd</a>                             |
| chier  | <a href="http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/classhierarchy.xsd">http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/classhierarchy.xsd</a> |

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

314 **Table 4 – XSD DSP Numbers**

| XSD File Name   | DSP Number |
|---|------------|
| <a href="http://schemas.dmtf.org/wbem/wscim/X/common.xsd">http://schemas.dmtf.org/wbem/wscim/X/common.xsd</a>                                   | DSP8004    |
| <a href="http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/qualifiers.xsd">http://schemas.dmtf.org/wbem/wscim/X/cim-schema/Y/qualifiers.xsd</a> | DSP8005    |
| <a href="http://schemas.dmtf.org/wbem/wscim/X/classhiertype.xsd">http://schemas.dmtf.org/wbem/wscim/X/classhiertype.xsd</a>                     | DSP8006    |

## 315 7 Dereferencing Schema URI Locations in Order to Access XML 316 Schema

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

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

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

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

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

329 EXAMPLE:

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

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

336 EXAMPLE:

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

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

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

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

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

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

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

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

### 349 7.1.1 Validation of CIM Instances

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

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

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

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

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

## 367 8 Mapping Primitive Datatypes

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

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

373 **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<br>xs:date<br>xs:time<br>xs:dateTime<br>xs:string<br><br>Depending on use-case<br>See 8.1. | cim:cimDateTime              |
| char16       | xs:string<br><br>With maxLength restriction = 1  | cim:cimChar16                |
| <class> REF  | N/A  | cim:cimReference<br>See 8.2. |

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

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

- 377 Octetstring
- 378 EmbeddedInstance
- 379 EmbeddedObject

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

### 382 8.1 cimDateTime Datatype

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

```

384 <xs:complexType name="cimDateTime">
385   <xs:choice>
386     <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
387     <xs:element name="Interval" type="xs:duration"/>
388     <xs:element name="Date" type="xs:date"/>
389     <xs:element name="Time" type="xs:time"/>
390     <xs:element name="Datetime" type="xs:dateTime"/>
391   </xs:choice>
392 <xs:anyAttribute namespace="##any" processContents="lax"/>
393 </xs:complexType>
    
```

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

395 **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            |

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

398 **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.  |

## 399 8.2 CIM References

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

```

401 <xs:complexType name="cimReference">
402   <xs:sequence>
403     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
404   </xs:sequence>
405 <xs:anyAttribute namespace="##any" processContents="lax"/>
406 </xs:complexType>

```

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

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

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

```

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

```

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

```

423 <AssociatedComponent
424   xmlns:wsa="http://www.w3.org/2005/08/addressing">
425   <wsa:Address>. . .</wsa:Address>
426   . . . <!-- Other EPR elements as defined in the 2005/08 specification -->
427 </AssociatedComponent>
428

```

```

429 <AssociatedComponent
430   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
431   <wsa:Address>. . .</wsa:Address>
432   . . . <!-- Other EPR elements as defined in the 2004/08 specification -->
433 </AssociatedComponent>

```



### 434 8.3 `cimAnySimpleType` Datatype

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

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

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

### 454 8.4 Derivation by Restriction

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

```
457 . . .
458 <this:Name xmlns:AdditionalAttribute=". . . ">
459     myName
460 </this:Name>
461 . . .
```

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

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

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

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

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

### 473 8.5 WS-CIM Canonical Values

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

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

## 480 9 CIM Class to XML Schema Mappings

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

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

### 486 9.1 Class Namespace

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

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

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

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

493 Where:

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

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

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

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

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

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

506 `http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM\_ComputerSystem`

### 507 9.2 Properties

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

#### 511 9.2.1 General Principles

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

514 **9.2.1.1 General Rules**

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

- 516 • Every property of a CIM class shall be represented by a global element definition. Note that this  
 517 rule applies to properties locally defined on the class itself and properties inherited from  
 518 superclasses (see 9.4). The GED that corresponds to a CIM property shall exhibit the following  
 519 features:
- 520 – The GED shall be defined in the namespace of the class in the XML Schema (see 9.1).
  - 521 – The name of the GED shall be the same as the name of the CIM property.
  - 522 – The type of the GED shall comply with the datatype conversion table defined in clause 8.  
 523 However, in some cases, depending on what qualifiers apply to the CIM property, it is  
 524 necessary to restrict the default type of the GED element. The complete specification of the  
 525 type of the GED shall comply with the normative rules for mapping qualifiers. (See 11.2 for  
 526 the normative rules for mapping specific qualifiers to XSD structures.)

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

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

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

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

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

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

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

```
554 class EX_BaseComponent {
555     datetime InstallDate;
556     [ ...
557         Required, MaxLen ( 1024 ) ]
558     string Name;
559     string StatusDescriptions[];
560     string HealthStatus;
561 };
```

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

```
564 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true" />
565 <xs:element name="Name">
566   <xs:complexType>
567     <xs:simpleContent>
568       <xs:restriction base="cim:cimString">
569         <xs:maxLength value="1024" />
570         <xs:anyAttribute namespace="##any" processContents="lax" />
571       </xs:restriction>
572     </xs:simpleContent>
573   </xs:complexType>
574 </xs:element>
```

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

### 577 9.2.1.2 Runtime Rules for Attribute Value Assignment

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

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

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

```
588 class EX_SomeClass {
589   ...
590   [... Key]
591   string Name;
592 }
```

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

```
594 <EX_SomeClass>
595   ...
596   <Name cim:Key="true">MyName</Name>
597   ...
598 </EX_SomeClass>
```

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

## 600 9.2.2 Array Properties

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

### 602 9.2.2.1 General Rules

603 The rule for representing arrays is as follows:

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

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

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

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

```
613 [ ...
614     ArrayType ( "Indexed" ) ]
615 string StatusDescriptions[];
```

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

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

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

```
620 <EX_BaseComponent>
621   ...
622   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
623   <StatusDescriptions>AnotherStatusDescription</StatusDescriptions>
624   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
625   ...
626 </EX_BaseComponent>
```

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

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

```
630 [ ...
631     MaxLen ( 64 ) ]
632 string StatusDescriptions[];
```

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

```
636 <xs:element name="StatusDescriptions" nillable="true">
637   <xs:complexType>
638     <xs:restriction base="cim:cimString">
639       <xs:maxLength value="64"/>
640       <xs:anyAttribute namespace="##any" processContents="lax"/>
641     </xs:restriction>
642   </xs:complexType>
643 </xs:element>
```

### 644 9.2.2.2 Runtime Rules for Arrays

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

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

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

```
655 <EX_BaseComponent>
656   ...
657   <StatusDescriptions>SomeStatusDescription</StatusDescriptions>
658   <StatusDescriptions xsi:nil="true" />
659   <StatusDescriptions>AThirdStatusDescription</StatusDescriptions>
660   ...
661 </EX_BaseComponent>
```

### 662 9.2.3 Properties with a ValueMap Qualifier

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

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

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

- 669 • If the `ValueMap` consists of only discrete values, the `ValueMap` shall be mapped to a  
670 `<xs:restriction>` consisting of an enumeration as follows:
  - 671 – The base type of the restriction shall be the WS-CIM datatype corresponding to the CIM  
672 datatype of the property (see 8).
  - 673 – Each discrete value of the `ValueMap` shall be mapped to a corresponding  
674 `<xs:enumeration>` element within the restriction.
- 675 • If the `ValueMap` contains a value specifying a range, the whole `ValueMap` shall be mapped to a  
676 `<xs:restriction>` consisting of a `<xs:union>` as follows:
  - 677 – The base type of the restriction shall be `cim:cimAnySimpleType` (see 8.3).
  - 678 – The elements of the `<xs:union>` shall be determined according to the following rules:
    - 679 • Discrete values shall be mapped to elements to an `<xs:restriction>` as  
680 described in the first rule with the exception that the base type of the restriction shall  
681 be the corresponding base XSD type of the CIM datatype of the property (see  
682 clause 8);
    - 683 • Bounded ranges ( $m..n$ ) shall be mapped to an `<xs:restriction>` consisting of  
684 `<xs:minInclusive="m">` and `<xs:maxInclusive="n">` where the restriction  
685 base type shall be the corresponding base XSD type of the CIM datatype of the  
686 property;
    - 687 • Unbounded ranges open on the left ( $...n$ ) shall be mapped to an  
688 `<xs:restriction>` consisting of `<xs:maxInclusive="n">` where the restriction  
689 base type shall be the corresponding base XSD type of the CIM datatype of the  
690 property;
    - 691 • Unbounded ranges open on the right ( $m...$ ) shall be mapped to an  
692 `<xs:restriction>` consisting of `<xs:minInclusive="m">` where the restriction  
693 base type shall be the corresponding base XSD type of the CIM datatype of the  
694 property;
    - 695 • Open ranges ( $..$ ) shall be mapped to an `<xs:union>` consisting of all discrete values  
696 and/or ranges that are unclaimed by the other values and ranges in the `ValueMap` by  
697 applying the preceding rules for constructing the elements of the `<xs:union>`  
698 recursively.

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

```
700 [ ...
701     ValueMap { "OK", "Error", "Unknown" } ]
702 string HealthStatus;
```

703 The HealthStatus property is therefore mapped as follows:

```
704 <xs:element name="HealthStatus" nillable="true">
705     <xs:complexType>
706         <xs:simpleContent>
707             <xs:restriction base="cim:cimString">
708                 <xs:enumeration value="OK"/>
709                 <xs:enumeration value="Error"/>
710                 <xs:enumeration value="Unknown"/>
711                 <xs:anyAttribute namespace="##any" processContents="lax"/>
712             </xs:restriction>
713         </xs:simpleContent>
714     </xs:complexType>
715 </xs:element>
```

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

```
717 [ ...
718 ValueMap { "0", "1", "2", "3..15999", "16000..65535" },
719     Values { "Unknown", "Other", "Not Applicable", "DMTF Reserved",
720             "Vendor Reserved" }}
721 uint16 PortType;
```

722 The PortType property is therefore mapped as follows:

```
723 <xs:element name="PortType" nillable="true">
724     <xs:complexType>
725         <xs:simpleContent>
726             <xs:restriction base="cim:cimAnySimpleType">
727                 <xs:simpleType>
728                     <xs:union>
729                         <xs:simpleType>
730                             <xs:restriction base="xs:unsignedShort">
731                                 <xs:enumeration value="0"/>
732                                 <xs:enumeration value="1"/>
733                                 <xs:enumeration value="2"/>
734                             </xs:restriction>
735                         </xs:simpleType>
736                         <xs:simpleType>
737                             <xs:restriction base="xs:unsignedShort">
738                                 <xs:minInclusive value="3"/>
739                                 <xs:maxInclusive value="15999"/>
740                             </xs:restriction>
741                         </xs:simpleType>
742                         <xs:simpleType>
743                             <xs:restriction base="xs:unsignedShort">
744                                 <xs:minInclusive value="16000"/>
745                                 <xs:maxInclusive value="65535"/>
746                             </xs:restriction>
747                         </xs:simpleType>
748                     </xs:union>
749                 </xs:simpleType>
750             <xs:anyAttribute namespace="##any" processContents="lax"/>
751         </xs:simpleContent>
752     </xs:complexType>
753 </xs:element>
```

```

751     </xs:restriction>
752   </xs:simpleContent>
753 </xs:complexType>
754 </xs:element>

```

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

```

757 [ ...
758 ValueMap { "1", "2", "3", "4", "5", "6", "7", "..", "16000.." },
759   Values { "Other", "Create", "Delete", "Detect", "Read", "Write",
760     "Execute", "DMTF Reserved", "Vendor Reserved" } ]
761 uint16 Activities;

```

762 The Activities property is therefore mapped as follows:

```

763 <xs:element name="Activities" nillable="true">
764   <xs:complexType>
765     <xs:simpleContent>
766       <xs:restriction base="cim:cimAnySimpleType">
767         <xs:simpleType>
768           <xs:union>
769             <xs:simpleType>
770               <xs:restriction base="xs:unsignedShort">
771                 <xs:enumeration value="1"/>
772                 <xs:enumeration value="2"/>
773                 <xs:enumeration value="3"/>
774                 <xs:enumeration value="4"/>
775                 <xs:enumeration value="5"/>
776                 <xs:enumeration value="6"/>
777                 <xs:enumeration value="7"/>
778               </xs:restriction>
779             </xs:simpleType>
780             <xs:simpleType>
781               <xs:union>
782                 <xs:simpleType>
783                   <xs:restriction base="xs:unsignedShort">
784                     <xs:enumeration value="0"/>
785                   </xs:restriction>
786                 </xs:simpleType>
787                 <xs:simpleType>
788                   <xs:restriction base="xs:unsignedShort">
789                     <xs:minInclusive value="8"/>
790                     <xs:maxInclusive value="15999"/>
791                   </xs:restriction>
792                 </xs:simpleType>
793               </xs:union>
794             </xs:simpleType>
795             <xs:simpleType>
796               <xs:restriction base="xs:unsignedShort">
797                 <xs:minInclusive value="16000"/>
798               </xs:restriction>
799             </xs:simpleType>
800           </xs:union>
801         </xs:simpleType>
802         <xs:anyAttribute namespace="##any" processContents="lax"/>
803       </xs:restriction>
804     </xs:simpleContent>
805   </xs:complexType>
806 </xs:element>

```



## 807 9.2.4 Octetstring Properties

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

### 811 9.2.4.1 General Rules

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

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

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

```
819 [...
820     Description ("The DER-encoded raw public key. " ),
821     OctetString ]
822 uint8 PublicKey[];
```

823 It would be represented by the following XSD:

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

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

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

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

```
828 [...
829     Description ("A CRL, or CertificateRevocationList, is a list of certificates which the "
830               "CertificateAuthority has revoked and which are not yet expired. " ),
831     Octetstring]
832 string CRL[];
```

833 It would be represented by the following XSD:

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

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

```
836 <CRL>0x000000F976H8A4...</CRL>
837 <CRL>0x000000C675D4G1...</CRL>
838 <CRL>0x000000D8B1H335...</CRL>
```

### 839 9.2.4.2 Runtime Value Conversion Rules

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

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

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

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

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

## 9.2.5 EmbeddedObject and EmbeddedInstance Properties

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

### 9.2.5.1 General Rules

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

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

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

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

It would have the following XSD representation:

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

### 9.2.5.2 Runtime Value Conversion Rules

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

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

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

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

```
class CIM_Part {
  string Label;
  int PartNo;
};
class CIM_Component {
  [Key]
```

```

895     string ID;
896     [EmbeddedInstance("CIM_Part")]
897     string Part;
898 };

```

899 Given an embedded instance of CIM\_Part with Label="Front Panel" and PartNo="19932", the  
900 following is a valid instance representation in the runtime XML instance document:

```

901 <CIM_Component xmlns="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Component"
902     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
903     <ID>ua123</ID>
904     <Part xmlns:e="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Part"
905         xsi:type="e:CIM_Part_Type">
906         <e:Label>Front Panel</e:Label>
907         <e:PartNo>19932</e:PartNo>
908     </Part>
909 </CIM_Component>

```

## 910 9.3 Class Structure

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

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

### 915 9.3.1 General Rules

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

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

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

```

925 Where:

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

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

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

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

- 940 • All elements that do not represent array properties shall have `minOccurs="0"` except for  
941 elements that correspond to properties that are designated with a `Key` or `Required` qualifier  
942 with an effective value of `true`. Elements that do not represent arrays and represent key and  
943 required properties shall have `minOccurs="1"`. Because 1 is the default value for  
944 `minOccurs`, it does not need to be explicitly expressed.
- 945 • All elements that represent array properties shall have `minOccurs="0"`. If the array size is  
946 specified in the CIM definition, the array property shall have `maxOccurs="array size"`. If  
947 the array size is not specified, the array property shall have `maxOccurs="unbounded"`.
- 948 • All elements except arrays shall have `maxOccurs="1"`. Because 1 is the default value for  
949 `maxOccurs`, it does not need to be explicitly expressed.
- 950 • Array properties (see 9.2.1.1) shall have a multiplicity that corresponds to the specification of  
951 their size in CIM. A bounded array in CIM shall be specified with a `maxOccurs` equal to the size  
952 of the array. If no size is specified in CIM, the schema element shall be specified with  
953 `minOccurs="0"` and `maxOccurs="unbounded"`.
- 954 • The schema of the CIM class shall support an open schema. Open schema means different  
955 protocols are able to add protocol-specific elements to instance documents.
  - 956 – To allow Open Content, the final element in the sequence shall be as follows:  
957 

```
<xs:any namespace="##other" processContents="lax" minOccurs="0"  
958     maxOccurs="unbounded" />
```
  - 959 – To allow attributes to be added to the element that represents the CIM class, following the  
960 sequence, the complex type shall allow any attribute to be added to the class with an  
961 `xs:anyAttribute` element, as follows:  
962 

```
<xs:anyAttribute namespace="##any" processContent="lax" />
```
- 963 • The class itself shall be represented by a GED of the type defined in the preceding rule. The  
964 name of this element shall be the name of the CIM class including its CIM schema name.

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

```
966 <xs:complexType name="EX_BaseComponent_Type">
967   <xs:sequence>
968     <xs:element ref="class:HealthStatus" minOccurs="0" />
969     <xs:element ref="class:InstallDate" minOccurs="0" />
970     <xs:element ref="class:Name" />
971     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded" />
972     <xs:any namespace="##other" processContents="lax" minOccurs="0"
973       maxOccurs="unbounded" />
974   </xs:sequence>
975   <xs:anyAttribute namespace="##any" processContent="lax" />
976 </xs:complexType>
977 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type" />
```

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

### 979 9.3.2 Runtime Property Value Rules

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

- 982 • For "get" operations, CIM service implementations returning the class's GEDs may omit  
983 schema-optional (`minOccurs="0"`) properties that have NULL values from responses. Clients  
984 are to interpret such omitted properties as having NULL values for those properties.
- 985 • Empty arrays (arrays with no members) shall not be included in responses to "get" requests. If  
986 the array is required, clients shall interpret the absence of all elements for the array to mean  
987 that the array is empty (no members). If the array is not required, clients shall interpret the  
988 absence of all elements for the array to mean that the array is either empty or NULL.
- 989 • A WS-CIM server shall return all current elements of an array.
- 990 • For "set" operations using the class's GEDs, clients may omit schema-optional  
991 (`minOccurs="0"`) properties. Service interpretation of the absence of such properties is  
992 protocol dependent.
- 993 • An instance document conformant to this specification shall include any elements from a foreign  
994 namespace at the end of the sorted list of CIM class elements. Elements from a foreign  
995 namespace may appear in any order.
- 996 • If a `Version` qualifier is associated with the CIM class, the `cim:Version` attribute may be  
997 applied to the element that represents the class in an XML instance document. Use of the  
998 `cim:Version` attribute shall conform to the following rule:  
  
999 If the attribute is present, its value shall be assigned as the value of the `Version` qualifier that  
1000 is associated with the CIM class, in the XML instance. If the CIM class does not have the  
1001 `Version` qualifier associated with it, this attribute shall be omitted.

1002 The MOF typically contains the `Version` qualifier, which specifies the latest version of the CIM class in  
1003 CIM:

```
1004 [ ... Version ( "2.10.2" ) ]  
1005 class EX_SomeClass {  
1006   ...  
1007 }
```

1008 The class may be represented in an instance document as follows:

```
1009 <EX_SomeClass cim:Version="2.10.2">  
1010   ...  
1011 </EX_SomeClass>
```

## 1012 9.4 Class Inheritance

1013 CIM inheritance is not modeled in the XML Schema of classes or within XML instances.

1014 Class inheritance is governed by the following rules:

- 1015 • Besides including the GEDs for the properties defined in a class (see 9.2.1.1), the namespace  
1016 for a class shall also include the GEDs for properties inherited from its superclasses. The class  
1017 type definition shall contain references to GEDs for all properties defined in and inherited into  
1018 the class according the rules in 9.3.1.
- 1019 • In general, classes inherit all properties specified in or inherited by their superclasses along with  
1020 all qualifiers that are specified as `ToSubClass`. However, properties with the same name may  
1021 be encountered within an inheritance chain. The `Override` qualifier determines special  
1022 behaviors that shall be observed by conversion algorithms when encountering properties with  
1023 duplicate names in the inheritance chain. The following rules govern the mapping of properties  
1024 with duplicate names:

- 1025           – When multiple properties in the inheritance chain that have the same name are not  
1026           overridden (that is, the effective value of the *Override* qualifier throughout the inheritance  
1027           chain is *NULL*), the property and its qualifiers in the most derived class shall be mapped.  
1028           All other duplicate named properties shall not be mapped.
- 1029           – When a property in a derived class overrides another property (of the same name and  
1030           type) in a superclass, the property in the most derived class shall be mapped, including all  
1031           qualifiers inherited from the overridden property. The overridden property shall not be  
1032           mapped.
- 1033           • The inheritance of qualifiers that pertain to properties shall comply with the inheritance rules  
1034           regarding qualifiers in C.3.
- 1035           • The definition of a derived class shall follow all other rules for constructing classes as defined in  
1036           9.3.1.

1037           NOTE: The metadata fragments for a property shall include any inherited qualifiers, subject to the qualifier inheritance  
1038           rules defined in 11.3. For more information about metadata fragments, see clause 11.

1039           Inheritance rests on the same type of examples presented in 9.2 and 9.3. The only addition is that the  
1040           properties inherited from a class's superclasses are included in the GEDs and class complex type  
1041           definition. For a complete example of inheritance, see C.2.2.

## 1042    **9.5 Method Parameters**

1043           The invocation of a CIM extrinsic method is represented by two separate messages:

- 1044           • the request input message, which represents the invocation of the method and the set of input  
1045           parameters
- 1046           • the response output message, which represents the output parameters and the method return  
1047           value in the successful case

1048           This clause specifies the XML Schema for these elements. These elements are then included as parts in  
1049           the WSDL input and output messages (see 10.1).

### 1050    **9.5.1 General Rules**

1051           The rules in this clause apply to mapping method input and output parameters and method return values.

1052           The GED used for the request input message is called the *input message GED*. The GED used for the  
1053           response output message is called the *output message GED*. The following rules specify the definition of  
1054           these GEDs:

- 1055           • The class namespace of the CIM class being mapped shall contain the input and output  
1056           message GEDs of all methods owned by the class and inherited from the superclasses. See  
1057           9.5.2, which defines class ownership of methods inherited from superclasses.
- 1058           • The names of these GEDs are defined by the following rules:
- 1059           – The *name* of the input message GED shall be the name of the CIM method with *\_INPUT*  
1060           appended.
- 1061           – The *name* of the output message GED shall be the name of the CIM method with *\_OUTPUT*  
1062           appended.
- 1063           • Each GED shall be defined as a complex type containing an *xs:sequence* of in-line elements  
1064           that represent the respective input or output parameters and return value of the CIM method as  
1065           immediate children. This structure is further defined in the rest of this clause.

1066 The following rules define the mapping of individual input and output parameters and the return value of  
1067 the CIM method, and the structure of the complex type used for defining the GEDs:

- 1068 • Input and output parameters of a CIM method shall be mapped to elements with the same  
1069 name as the corresponding parameter name. The following rules define the features of these  
1070 elements:
  - 1071 – The type of an element that represents an input or output parameter shall be mapped as  
1072 defined in clause 8.
  - 1073 – Parameters that are not qualified with a `Required` qualifier shall be mapped to elements  
1074 that contain the `nillable="true"` attribute.
- 1075 • The complex type used to define the type of the input message GED shall contain all and only  
1076 those elements that correspond to method parameters that have their `In` qualifier effectively  
1077 defined as `true`. The sequence of these elements in the complex type shall correspond to the  
1078 sequence of the input parameters in the CIM definition of the method.
 

1079 If the method has no input parameters, the complex type used in the GED shall be empty (that  
1080 is, `<xs:complexType/>`).
- 1081 • The complex type used to define the type of the output message GED shall contain all elements  
1082 that correspond to method parameters that have their `Out` qualifier effectively defined as `true`.  
1083 The sequence of these elements in the complex type shall correspond to the sequence of the  
1084 output parameters in the CIM definition of the method.
- 1085 • The complex type used to define the output message GED shall contain an element of  
1086 `name="ReturnValue"` as the final element in its sequence. The following rules govern the  
1087 structure of this element:
  - 1088 – The XSD type of this element shall be mapped from the CIM method type in compliance  
1089 with the datatype conversion defined in clause 8.
  - 1090 – The element shall include the attribute `nillable="true"`. (See the following note.)
- 1091 • Parameters and return values may be defined in CIM with `ValueMap` qualifiers. Mapping these  
1092 `ValueMaps` to metadata fragments is required (see 11.1). In addition, a `ValueMap` shall be  
1093 mapped to an enumeration/union associated with the mapped parameter or return value in the  
1094 XSD (see 11.2). The mapping shall conform to the rules for mapping `ValueMaps` described in  
1095 9.2.3.

1096 Notes on the preceding rules:

- 1097 • A parameter shall occur in both the complex types used to define the input and output message  
1098 GEDs if it has both the `In` and `Out` qualifiers effectively defined as `true`.

1099 [DSP0004](#) allows NULL as the default return value for all methods. Thus, this specification must allow for  
1100 the possibility that the return value of any method may be NULL.

## 1101 9.5.2 Inheritance of Methods

1102 Classes may inherit some of the methods that they own. In general, classes inherit all methods specified  
1103 in or inherited by their superclasses, along with all qualifiers that are specified as `ToSubClass`. The  
1104 `Override` qualifier, however, determines special behavioral considerations on the part of conversion  
1105 algorithms. The following rules govern the inheritance of methods under conditions of override:

- 1106 • When multiple methods in the inheritance chain that have the same name are not overridden,  
1107 the method in the most derived class shall be mapped. Any other duplicate, but not overridden,  
1108 methods shall not be mapped.

- 1109 • When a method in a derived class overrides another method (of the same name and signature)
- 1110 in a superclass, the method in the most derived class shall be mapped, including all qualifiers
- 1111 inherited from the overridden methods. The overridden method shall not be mapped.
- 1112 • The inheritance of qualifiers pertaining to methods shall comply with the inheritance rules
- 1113 regarding qualifiers in C.3.

1114 EXAMPLE: As an example of the preceding rules, consider the following MOF method definition extracted from the

1115 example in C.1.2. (See C.2.2 for the complete example.)

```
1116 class EX_DerivedComponent
1117 {
1118   ...
1119   uint32 RequestStateChange([IN] uint16 RequestedState, [OUT] [IN(False)] CIM_SomeClass REF
1120   ResultClass, [IN] datetime TimeoutPeriod);
1121 };
```

1122 The input parameters, designated in the MOF by the `In` qualifier, would be mapped as follows:

```
1123 <xs:element name="RequestStateChange_INPUT">
1124 <xs:complexType>
1125   <xs:sequence>
1126     <xs:element name="RequestedState" type="cim:cimUnsignedShort"
1127       nillable="true"/>
1128     <xs:element name="TimeoutPeriod" type="cim:cimDateTime"
1129       nillable="true"/>
1130   </xs:sequence>
1131 </xs:complexType>
1132 </xs:element>
```

1133 The output parameters, designated in the MOF by the `Out` qualifier, would be mapped in the following

1134 way. Note that the complex type includes an element that represents the return value of the CIM method.

```
1135 <xs:element name="RequestStateChange_OUTPUT">
1136 <xs:complexType>
1137   <xs:sequence>
1138     <xs:element name="ResultClass" type="cim:cimReference"
1139       nillable="true"/>
1140     <xs:element name="ReturnValue" type="cim:cimUnsignedInt"
1141       nillable="true"/>
1142   </xs:sequence>
1143 </xs:complexType>
1144 </xs:element>
```

### 1145 9.5.3 Parameters and Return Values with ValueMaps

1146 Input and output parameters and return values of the method may be specified with a `ValueMap` qualifier.

1147 The `ValueMap` shall be mapped to the XSD element that represents the parameter or return value.

1148 The `RequestedState` input parameter must be defined as an enumeration in accordance with its

1149 `ValueMap` (see C.1.2 for this `ValueMap`):

```
1150 <xs:element name="RequestedState" nillable="true">
1151 <xs:complexType>
1152 <xs:simpleContent>
1153 <xs:restriction base="cim:cimUnsignedShort">
1154 <xs:enumeration value="2"/>
1155 <xs:enumeration value="3"/>
1156 <xs:enumeration value="4"/>
```



```

1157     <xs:anyAttribute namespace="##any" processContents="lax"/>
1158     </xs:restriction>
1159     </xs:simpleContent>
1160     </xs:complexType>
1161 </xs:element>

```

## 1162 9.6 CIM Instances

1163 This clause describes the representation of CIM instances. The intended scope is all representations of  
 1164 CIM instances used in any protocols.

1165 CIM instances are represented according to the following rules:

- 1166 • Representations of CIM instances shall be XML instance documents that conform to the XSD  
 1167 schema for their CIM creation class, as defined in clause 9.
- 1168 • The class namespace used within an instance document shall be a namespace URI and it shall  
 1169 be defined as follows:

1170 [1172 Where:](http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-major-<br/>
  1171 version '/' cim-class-schema ' _ ' cim-class-name</a></p>
</div>
<div data-bbox=)

- 1173 – [wscim-major-version](#) is the major version number of this specification. Note that this  
 1174 version number changes only if there are incompatible changes in the specification.
- 1175 – [cim-schema-major-version](#) is the major version number of the CIM schema version to  
 1176 which the class being converted belongs. Note that this version number changes only if  
 1177 there are incompatible changes in the CIM schema.
- 1178 • The location of the specific schema used to construct the instance should be declared by use of  
 1179 the `xsi:schemaLocation` attribute where the namespace URI and the specific class schema  
 1180 document URI location are combined as the value of the attribute, separated by whitespace.  
 1181 This provides a means for a recipient of the instance to determine which version of CIM defines  
 1182 the structure of this instance.

1183 For example:

1184 If the instance document is constructed under CIM schema version 2.11.0 and WS-CIM  
 1185 mapping specification 1.3.0, the following `xsi:schemaLocation` attribute should be specified in  
 1186 the instance document (where *ClassName* is the name of the CIM class of the instance):

```

1187 xsi:schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/ClassName  

1188 http://schemas.dmtf.org/wbem/wscim/1.3.0/cim-schema/2.11.0/ClassName.xsd"

```

1189 If the URI in the attribute value can be de-referenced, then strict XML schema validation can be achieved.

## 1190 9.7 Superclass Class Representation

1191 The CIM Schema defines CIM classes in subclass / superclass relationships (hierarchies). For example,  
 1192 MOF reflects this structure in the definition of the class. A MOF statement of the following form defines  
 1193 `CIM_ClassA` as a subclass of the superclass `CIM_ClassB`:

```

1194 class CIM_ClassA : CIM_ClassB

```

1195 A MOF statement of the following form defines `CIM_ClassC` as a top-level class with no superclass:

```

1196 class CIM_ClassC

```

1197 Some management protocols may require a representation of the subclass/superclass hierarchy of a  
 1198 class as a value of an XML element in instance documents. The XSD mechanism by which to support  
 1199 representing this structure as a value of an XML element is provided in a separate schema that may be  
 1200 imported by protocols that require this capability for their instance documents.

1201 The immediate subclass/superclass relationship of a class shall be mapped to a single GED in the  
 1202 classhierarchy namespace according to the following rules:

- 1203 • Elements that describe the subclass / superclass relationships shall be placed in a separate  
 1204 schema from the WS-CIM class schema. For subclass / superclass relationships defined in the  
 1205 CIM Schema, the schema shall be named as follows:

1206 `'http://schemas.dmtf.org/wbem/wscim/' wscim-major-version '/cim-schema/' cim-schema-`  
 1207 `major-version '/classhierarchy'`

1208 Where:

- 1209 – `wscim-major-version` is the major version number of this specification. Note that this  
 1210 version number changes only if there are incompatible changes in the specification.
- 1211 – `cim-schema-major-version` is the major version number of the CIM schema version to  
 1212 which the class being converted belongs. Note that this version number changes only if  
 1213 there are incompatible changes in the CIM schema.

1214 This schema shall import the `http://schemas.dmtf.org/wbem/wscim/n/classhiertype` namespace, where '*n*'  
 1215 represents the version number of this namespace.

1216 The class being mapped shall be represented by a GED whose name is derived from the name of the  
 1217 CIM class, appended with "`_Class`". This element shall be defined by an anonymous complex type that  
 1218 follows one of the following patterns:

- 1219 • The WS-CIM schema of a top-level class, `XXX_ClassC`, shall define the `_Class` element as a  
 1220 restriction of the `ctype:ClassHierarchyType`. The following pattern illustrates this rule:

```
1221 <xs:element name="XXX_ClassC_Class">
1222   <xs:complexType>
1223     <xs:complexContent>
1224       <xs:restriction base="ctype:ClassHierarchyType" />
1225     </xs:complexContent>
1226   </xs:complexType>
1227 </xs:element>
```

- 1228 • The WS-CIM schema of a subclass, `XXX_ClassA`, that is derived by a superclass,  
 1229 `XXX_ClassB`, shall restrict the `ctype:classHierarchyType` to contain a single element that  
 1230 references the corresponding `_Class` GED of the superclass. The following pattern illustrates  
 1231 this rule:

```
1232 <xs:element name="XXX_ClassA_Class">
1233   <xs:complexType>
1234     <xs:complexContent>
1235       <xs:restriction base="ctype:ClassHierarchyType">
1236         <xs:sequence>
1237           <xs:element ref="chier:XXX_ClassB_Class" />
1238         </xs:sequence>
1239       </xs:restriction>
1240     </xs:complexContent>
1241   </xs:complexType>
1242 </xs:element>
```

1243 See C.2.4 for an example classhierarchy schema.

## 1244 10 CIM Methods to WSDL Mappings

1245 This clause defines the structures that are necessary to define the messages and operation structures  
1246 required for mapping a CIM method to WSDL.

### 1247 10.1 Defining WSDL Message Structures

1248 This clause provides the rules for creating WSDL message structures.

1249 The rules that govern the creation of WSDL message structures for a method are as follows:

- 1250 • The name of the WSDL input message should be the name of the CIM method plus  
1251 `_InputMessage`. The following rules specify the structure of this element:
  - 1252 – The name of the `wsdl:part` element should be "body".
  - 1253 – The `element` attribute of the `wsdl:part` element shall specify the QName of the input  
1254 message GED for the CIM method (see 9.5).
- 1255 • The name of the WSDL output message should be the name of the CIM method plus  
1256 `_OutputMessage`. The following rules specify the structure of this element:
  - 1257 – The name of the `wsdl:part` element should be "body".
  - 1258 – The `element` attribute of the `wsdl:part` element shall specify the QName of the output  
1259 message GED defined for the CIM method (see 9.5).

1260 **EXAMPLE:** The `wsdl:message` elements for the `RequestStateChange` CIM method (see 9.5) would  
1261 be specified in the WSDL document as follows. The `wsdl:message` intended for input to the WSDL  
1262 operation would be as follows (where the "class:" namespace prefix represents the namespace of the  
1263 class whose interface is being exposed through this WSDL):

```
1264 <wsdl:message name="RequestStateChange_InputMessage">
1265   <wsdl:part name="body"
1266     element="class:RequestStateChange_INPUT" />
1267 </wsdl:message>
```

1268 See C.3 for an example that shows the complete specification of the WSDL messages for this operation.

### 1269 10.2 Defining WSDL Operation Structures

1270 This specification defines *only* the structure of WSDL `portType` operations.

1271 The rules governing the structure of the WSDL operations used to invoke CIM methods are as follows:

- 1272 • The `name` attribute of the `wsdl:operation` element shall be the name of the CIM method.
- 1273 • The `name` attributes of the `wsdl:input` and `wsdl:output` child elements should be the name  
1274 of the `wsdl:messages` that are referenced by these elements.
- 1275 • The `message` attributes of the `wsdl:input` and `wsdl:output` elements shall specify the  
1276 QName of input and output message elements defined in 10.1.

1277 **EXAMPLE:** The `RequestStateChange` CIM method (see 9.5) is defined as follows:

```
1278 <wsdl:definitions
1279   ...
1280   xmlns:thisWSDL="http://. . .wsdl"
```

```

1281     ...>
1282 <wsdl:operation name="RequestStateChange">
1283   <wsdl:input name="RequestStateChange_InputMessage"
1284     message="thisWSDL:RequestStateChange_InputMessage" />
1285   <wsdl:output name="RequestStateChange_OutputMessage"
1286     message="thisWSDL:RequestStateChange_OutputMessage" />
1287 </wsdl:operation>
1288 </wsdl:definitions>

```

1289 This definition should be included in the `wsdl:portType` section of a WSDL document of a service that  
 1290 makes the CIM `RequestStateChange` method available to clients.

### 1291 10.3 Defining `wsa:Actions`

1292 The Addressing specifications ([Web Services Addressing \(WS-Addressing\) 1.0 – Core](#) and [DSP0226](#),  
 1293 "Management Addressing" clause) define the information model and syntax for the abstract messaging  
 1294 property Action. This property is defined as an IRI ([RFC 3987](#)), which identifies the semantics implied by  
 1295 a message (input, output, or fault). For the purposes of this specification, the Action property is restricted  
 1296 to a URI ([RFC 3986](#)) (a sequence of characters from a limited subset of the repertoire of US-ASCII  
 1297 characters).

1298 The details of how the action URI is specified on description artifacts are left to the specific protocol-  
 1299 binding specifications. Action URIs for faults are always left to the protocol-binding specifications.

1300 The rules for constructing WSA action URIs for input and output operation elements are as follows:

- 1301 • The action URI for an input message shall have the following form:

1302 class-namespace-name '/' input-name

1303 Where:

- 1304 – class-namespace-name is the full namespace name of the class being mapped as defined  
 1305 in 9.1.
- 1306 – input-name is the name of the CIM method.

- 1307 • The action URI for an output message shall have the following form:

1308 class-namespace-name '/' output-name 'Response'

1309 Where:

- 1310 – class-namespace-name is the full namespace name of the class being mapped as defined  
 1311 in 9.1.
- 1312 – output-name is the name of the output CIM method.

1313 EXAMPLE: The action URI for the input message of the `RequestStateChange` method is as follows:

```

1314 wsa:Action="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/EX_DerivedComponent/
1315 RequestStateChange"

```

1316 The action URI for the output message of the `RequestStateChange` method is as follows:

```

1317     wsa:Action="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent/
1318     RequestStateChangeResponse"

```

1319 See C.3 for an example that shows a complete mapping of the `RequestStateChange` method.

## 1320 11 Qualifier Mappings

1321 This clause defines the mapping of qualifiers to metadata fragments. The definition of the container for  
1322 metadata fragments is left to the specific protocols to specify.

### 1323 11.1 General Format

1324 Rules for mapping qualifiers fall into two categories:

- 1325 • rules that describe the type definitions of qualifiers
- 1326 • rules that describe the XSD elements to which qualifiers are mapped

1327 This specification provides type definitions for the qualifier types used in CIM in common.xsd and  
1328 mappings for all CIM qualifiers in the qualifiers.xsd (Annex A.2). It is expected that user-defined qualifiers  
1329 follow the mapping rules for mapping qualifiers described in the following clauses.

1330 In addition to the mapping rules, user-defined qualifier mappings are governed by the following rules:

- 1331 • User-defined qualifiers shall not use any qualifier namespace defined in this specification.

1332 The qualifier types (types with names of the qualifier *Type*) defined in common.xsd should be used to  
1333 define user-defined qualifiers. In the majority of cases, it is not necessary to create a new type definition  
1334 to define a qualifier.

#### 1335 11.1.1 Single-Valued Qualifiers

1336 This clause describes the rules for mapping single-valued qualifiers.

1337 Single-valued qualifiers that have an effective value that matches their default value shall not be mapped  
1338 to a metadata fragment.

1339 The rules for mapping single-valued qualifiers that have an effective value that is not their global default  
1340 value are as follows:

- 1341 • The rules for defining the type of a single-valued qualifier are as follows:
  - 1342 – Single-valued qualifiers shall be mapped to a complex type that is an extension of a simple  
1343 content.
  - 1344 – The base type of this complex type shall correspond to the type of qualifier according to the  
1345 datatype conversion rules defined in clause 8.
  - 1346 – The complex type shall extend the base type with a Boolean attribute of the name  
1347 `qualifier`. This attribute is defined in the `cim` namespace. This attribute shall be  
1348 specified with the XML Schema attribute `use="required"`.
- 1349 • The rules for mapping a single-valued qualifier are as follows:
  - 1350 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of  
1351 the qualifier and which has been defined by the preceding rule. The name of the element  
1352 shall be the name of the qualifier.
  - 1353 – The value of the `cim:qualifier` attribute in the mapped metadata fragment shall be `true`  
1354 in all mappings.
  - 1355 – Single-valued qualifiers implicitly have the multiplicity `minOccurs="0" maxOccurs="1"`.  
1356 Protocols should provide a mechanism by which to express this multiplicity in the schema  
1357 of the document that contains generated metadata fragments.

- 1358 • The format of a schema element for defining a single-valued qualifier is as follows:

```
1359 '<xs:element name="" cim:qualifier-name ' ' type="" qualifier-type ' ' />''
```

1360 Where:

- 1361 – qualifier-type is typically one of the qualifier *Type* types defined in common.xsd (but may be
- 1362 a user-defined type that complies with the mapping rules).

1363 cim:qualifier-name is the name of the qualifier, qualified by its namespace prefix.

1364 EXAMPLE: A generalized example of the mapping of a single-valued qualifier is as follows:

```
1365 <ns:QualifierName cim:qualifier="true">
```

```
1366     value
```

```
1367 </ns:QualifierName>
```

1368 Where:

- 1369 – QualifierName is the name of the qualifier.
- 1370 – ns is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1371 – value is the string representation of the qualifier value.

1372 For example, the mapping of the CIM qualifier `Abstract` is as follows:

```
1373 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
```

### 1374 11.1.2 Multi-Valued Qualifiers

1375 This clause describes the rules for mapping multi-valued qualifiers.

1376 Multi-valued qualifiers are qualifiers that are arrays. Multi-valued qualifiers that have an effective value  
1377 that matches their default value shall not be mapped to a metadata fragment.

1378 The rules for mapping multi-valued qualifiers that have an effective value that is not their global default  
1379 value are as follows:

- 1380 • The rules for defining the type of a multi-valued qualifier are as follows:
  - 1381 – Multi-valued qualifiers shall be mapped to a complex type that is an extension of a complex
  - 1382 content.
  - 1383 – The base type of this complex type shall correspond to the single-valued qualifier *Type* of
  - 1384 the member elements from which the qualifier array is constructed.
- 1385 • The rules for mapping a multi-valued qualifier are as follows:
  - 1386 – The qualifier shall be mapped to a GED whose type corresponds to the datatype type of
  - 1387 the qualifier and which has been defined by the preceding rule. The `name` of the element
  - 1388 shall be the same as that of the qualifier itself.
  - 1389 – The value of the `cim:qualifier` attribute in the metadata fragments shall be `true` in all
  - 1390 mappings.
  - 1391 – Multi-valued qualifiers implicitly have the default multiplicity `minOccurs="0"`
  - 1392 `maxOccurs="unbounded"`. Qualifier declarations may explicitly set different bounds on
  - 1393 an array qualifier. Protocols should provide a mechanism by which to express the size of
  - 1394 an array qualifier in the schema of the document that contains generated metadata
  - 1395 fragments. The `minOccurs` and `maxOccurs` values shall correspond either to the default
  - 1396 values if no qualifier array size is declared or to the declared qualifier array size.

1397 NOTE: The common.xsd file defines a string array of qualifier values, `qualifierSArray`. This definition complies  
1398 with the first mapping rule in this clause. Therefore, in the majority of cases, it is not necessary to create a new array

1399 complex type definition to define a multi-valued qualifier. Rather, it is sufficient to use the `qualifierSArray` type  
 1400 defined in `common.xsd`.

1401 The format for a schema for defining a multi-valued qualifier is as follows:

```
1402 <xs:element name="cim:qualifier-name" type="qualifier-array-type"/>
```

1403 Where:

- 1404 – `cim:qualifier-name` is the name of the qualifier, qualified by its namespace prefix.
- 1405 – `qualifier-array-type` is typically the `qualifierSArray` type defined in the `common.xsd` file  
 1406 (but may be a user-defined type that complies with the mapping rules).

1407 **EXAMPLE 1:** A generalized example of the mapping of a multi-valued qualifier is as follows:

```
1408 <ns:QualifierName cim:qualifier="true">
1409     value
1410 </ns:QualifierName>
1411 ... // repeat QualifierName elements for each member of the array
```

1412 Where:

- 1413 – `QualifierName` is the name of the qualifier.
- 1414 – `ns` is the namespace prefix referencing the namespace in which this qualifier is defined.
- 1415 – `n` is a sequential integer that represents the position of the entry in the array.
- 1416 – `value` is the string representation of the qualifier value.

1417 **EXAMPLE 2:** For example, the mapping of a `ValueMap` qualifier containing three entries ( "OK", "Error",  
 1418 "Unknown" ) is as follows:

```
1419 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
1420 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
1421 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
```

1422 A complete mapping of all qualifiers is provided in `qualifiers.xsd` (Annex A.2).

## 1423 11.2 Mapping CIM Qualifiers to XSD Elements

1424 All qualifiers are mapped using the normative rules in 11.1. The qualifiers listed in Table 8 are also  
 1425 mapped directly into XSD features.

1426 **Table 8 – CIM Qualifiers Mapped to XSD Elements**

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

| CIM Qualifier | MOF Example        | Mapped to XSD Structure  |
|---------------|--------------------|--|
| MaxLen        | [MaxLen ( 1024 )]  | <p>Mapped to a restriction using <code>xs:maxLength</code> 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="716 411 1421 699">&lt;xs:element name="PropName"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:restriction base="cim:cimString"&gt;         &lt;xs:maxLength value="1024"/&gt;       &lt;xs:anyAttribute .../&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleContent&gt; &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>  |
| MaxValue      | [MaxValue ( 100 )] | <p>Mapped to a restriction using <code>xs:maxInclusive</code> on an integer datatype. Required, with the following exception:</p> <p>A qualifier value of NULL, which indicates the largest value allowed by the type, should not be mapped.</p> <p>For example:</p> <pre data-bbox="716 905 1421 1186">&lt;xs:element name="PropName"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:restriction base="cim:cimUnsignedShort"&gt;         &lt;xs:maxInclusive value="100"/&gt;       &lt;xs:anyAttribute ... /&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleContent&gt; &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre> |
| MinLen        | [MinLen ( 10 )]    | <p>Mapped to a restriction using <code>xs:minLength</code> on a string datatype. Required, with the following exception:</p> <p>A qualifier value of 0 should not be mapped.</p> <p>For example:</p> <pre data-bbox="716 1367 1421 1648">&lt;xs:element name="PropName"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:restriction base="cim:cimString"&gt;         &lt;xs:minLength value="10"/&gt;       &lt;xs:anyAttribute ... /&gt;     &lt;/xs:restriction&gt;   &lt;/xs:simpleContent&gt; &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>   |



| CIM Qualifier                      | MOF Example               | Mapped to XSD Structure  |
|------------------------------------|---------------------------|--|
| 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 <code>NULL</code>, which indicates the smallest value allowed by the type, should not be mapped.</p> <p>For example:</p> <pre>&lt;xs:element name="PropName"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:restriction base="cim:cimUnsignedShort"&gt;         &lt;xs:minInclusive value="10"/&gt;         &lt;xs:anyAttribute . . ./&gt;       &lt;/xs:restriction&gt;     &lt;/xs:simpleContent&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre> |
| OctetString<br>uint8[]<br>string[] | [OctetString]             | <p>Normatively defined in 9.2.4.1</p> <p><code>cim:cimBase64Binary</code><br/><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 style="list-style-type: none"> <li>• For non-overridden properties, <code>Override(NULL)</code>, only the qualifiers in the most derived class shall be mapped.</li> <li>• For overridden properties, the effective values of inherited qualifiers shall be considered in the mapping.</li> </ul>    |
| 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 <code>false</code>, then <code>nillable="true"</code> (required).</p>  |
| ValueMap                           | [ValueMap (...)]          | <p>ValueMaps may be mapped to XSD as an enumeration (see 9.2.3).</p>   |

1427 **11.3 Inheritance of Qualifiers**

1428 In addition to inheritance of properties, references, and methods through class inheritance, qualifier  
 1429 values on any CIM elements are inherited. However, qualifiers are subject to special rules of inheritance.  
 1430 Qualifier inheritance behavior is defined by the Flavors associated with a particular qualifier declaration.

1431 The rules covering qualifier inheritance are summarized in the third column of Table 9. In Table 9, the  
 1432 term "overriding CIM elements in any subclasses" refers to CIM properties, references, and methods that  
 1433 override other occurrences of the properties, references, or methods in their superclasses and therefore  
 1434 form an inheritance chain. Note that duplicate property, reference, or method names that are *not*  
 1435 overridden interrupt the inheritance chain for these CIM elements to their superclasses.

1436

**Table 9 – Rules of Qualifier Inheritance**

| FLAVOR                      | Qualifier Inheritance Behavior (Informative)  | Metadata Fragment Mapping Behavior   |
|-----------------------------|---|--|
| Restricted                  | <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: <i>Abstract</i></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: <i>MaxLen</i></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: <i>Key</i></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

1437  
1438  
1439  
1440

1441 This annex provides examples of the WS-CIM Schema (DSP8004), the Qualifiers Schema (DSP8005),  
1442 and the Class Hierarchy Type Schema (DSP8006).

#### 1443 **A.1 Common WS-CIM Schema: DSP8004**

1444 This schema contains common definitions.

```

1445 <?xml version="1.0" encoding="utf-8" ?>
1446 <xs:schema targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1447     xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1448     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1449     elementFormDefault="qualified">
1450
1451 <!-- The following are runtime attribute definitions -->
1452     <xs:attribute name="Key" type="xs:boolean"/>
1453
1454     <xs:attribute name="Version" type="xs:string"/>
1455
1456 <!-- The following section defines the extended WS-CIM datatypes -->
1457
1458     <xs:complexType name="cimDateTime">
1459         <xs:choice>
1460             <xs:element name="CIM_DateTime" type="xs:string" nillable="true"/>
1461             <xs:element name="Interval" type="xs:duration"/>
1462             <xs:element name="Date" type="xs:date" />
1463             <xs:element name="Time" type="xs:time" />
1464             <xs:element name="Datetime" type="xs:dateTime"/>
1465         </xs:choice>
1466         <xs:anyAttribute namespace="##any" processContents="lax"/>
1467     </xs:complexType>
1468
1469     <xs:complexType name="cimUnsignedByte">
1470         <xs:simpleContent>
1471             <xs:extension base="xs:unsignedByte">
1472                 <xs:anyAttribute namespace="##any" processContents="lax"/>
1473             </xs:extension>
1474         </xs:simpleContent>
1475     </xs:complexType>
1476
1477     <xs:complexType name="cimByte">
1478         <xs:simpleContent>
1479             <xs:extension base="xs:byte">
1480                 <xs:anyAttribute namespace="##any" processContents="lax"/>
1481             </xs:extension>
1482         </xs:simpleContent>
1483     </xs:complexType>
1484
1485     <xs:complexType name="cimUnsignedShort">
1486         <xs:simpleContent>
1487             <xs:extension base="xs:unsignedShort">

```

```

1488     <xs:anyAttribute namespace="##any" processContents="lax"/>
1489   </xs:extension>
1490 </xs:simpleContent>
1491 </xs:complexType>
1492
1493 <xs:complexType name="cimShort">
1494   <xs:simpleContent>
1495     <xs:extension base="xs:short">
1496       <xs:anyAttribute namespace="##any" processContents="lax"/>
1497     </xs:extension>
1498   </xs:simpleContent>
1499 </xs:complexType>
1500
1501 <xs:complexType name="cimUnsignedInt">
1502   <xs:simpleContent>
1503     <xs:extension base="xs:unsignedInt">
1504       <xs:anyAttribute namespace="##any" processContents="lax"/>
1505     </xs:extension>
1506   </xs:simpleContent>
1507 </xs:complexType>
1508
1509 <xs:complexType name="cimInt">
1510   <xs:simpleContent>
1511     <xs:extension base="xs:int">
1512       <xs:anyAttribute namespace="##any" processContents="lax"/>
1513     </xs:extension>
1514   </xs:simpleContent>
1515 </xs:complexType>
1516
1517 <xs:complexType name="cimUnsignedLong">
1518   <xs:simpleContent>
1519     <xs:extension base="xs:unsignedLong">
1520       <xs:anyAttribute namespace="##any" processContents="lax"/>
1521     </xs:extension>
1522   </xs:simpleContent>
1523 </xs:complexType>
1524
1525 <xs:complexType name="cimLong">
1526   <xs:simpleContent>
1527     <xs:extension base="xs:long">
1528       <xs:anyAttribute namespace="##any" processContents="lax"/>
1529     </xs:extension>
1530   </xs:simpleContent>
1531 </xs:complexType>
1532
1533 <xs:complexType name="cimString">
1534   <xs:simpleContent>
1535     <xs:extension base="xs:string">
1536       <xs:anyAttribute namespace="##any" processContents="lax"/>
1537     </xs:extension>
1538   </xs:simpleContent>
1539 </xs:complexType>
1540
1541 <xs:complexType name="cimBoolean">
1542   <xs:simpleContent>
1543     <xs:extension base="xs:boolean">
1544       <xs:anyAttribute namespace="##any" processContents="lax"/>
1545     </xs:extension>

```

```

1546     </xs:simpleContent>
1547 </xs:complexType>
1548
1549 <xs:complexType name="cimFloat">
1550   <xs:simpleContent>
1551     <xs:extension base="xs:float">
1552       <xs:anyAttribute namespace="##any" processContents="lax"/>
1553     </xs:extension>
1554   </xs:simpleContent>
1555 </xs:complexType>
1556
1557 <xs:complexType name="cimDouble">
1558   <xs:simpleContent>
1559     <xs:extension base="xs:double">
1560       <xs:anyAttribute namespace="##any" processContents="lax"/>
1561     </xs:extension>
1562   </xs:simpleContent>
1563 </xs:complexType>
1564
1565 <xs:complexType name="cimChar16">
1566   <xs:simpleContent>
1567     <xs:restriction base="cim:cimString">
1568       <xs:maxLength value="1"/>
1569       <xs:anyAttribute namespace="##any" processContents="lax"/>
1570     </xs:restriction>
1571   </xs:simpleContent>
1572 </xs:complexType>
1573
1574 <xs:complexType name="cimBase64Binary">
1575   <xs:simpleContent>
1576     <xs:extension base="xs:base64Binary">
1577       <xs:anyAttribute namespace="##any" processContents="lax"/>
1578     </xs:extension>
1579   </xs:simpleContent>
1580 </xs:complexType>
1581
1582 <xs:complexType name="cimHexBinary">
1583   <xs:simpleContent>
1584     <xs:extension base="xs:hexBinary">
1585       <xs:anyAttribute namespace="##any" processContents="lax"/>
1586     </xs:extension>
1587   </xs:simpleContent>
1588 </xs:complexType>
1589
1590 <xs:complexType name="cimAnySimpleType">
1591   <xs:simpleContent>
1592     <xs:extension base="xs:anySimpleType">
1593       <xs:anyAttribute namespace="##any" processContents="lax"/>
1594     </xs:extension>
1595   </xs:simpleContent>
1596 </xs:complexType>
1597
1598 <xs:complexType name="cimReference">
1599   <xs:sequence>
1600     <xs:any namespace="##other" maxOccurs="unbounded" processContents="lax"/>
1601   </xs:sequence>
1602   <xs:anyAttribute namespace="##any" processContents="lax"/>
1603 </xs:complexType>

```

```

1604
1605 <!-- The following datatypes are used exclusively to define metadata fragments -->
1606 <xs:attribute name="qualifier" type="xs:boolean"/>
1607
1608 <xs:complexType name="qualifierString">
1609 <xs:simpleContent>
1610 <xs:extension base="cim:cimString">
1611 <xs:attribute ref="cim:qualifier" use="required"/>
1612 </xs:extension>
1613 </xs:simpleContent>
1614 </xs:complexType>
1615
1616 <xs:complexType name="qualifierBoolean">
1617 <xs:simpleContent>
1618 <xs:extension base="cim:cimBoolean">
1619 <xs:attribute ref="cim:qualifier" use="required"/>
1620 </xs:extension>
1621 </xs:simpleContent>
1622 </xs:complexType>
1623
1624 <xs:complexType name="qualifierUInt32">
1625 <xs:simpleContent>
1626 <xs:extension base="cim:cimUnsignedInt">
1627 <xs:attribute ref="cim:qualifier" use="required"/>
1628 </xs:extension>
1629 </xs:simpleContent>
1630 </xs:complexType>
1631
1632 <xs:complexType name="qualifierSInt64">
1633 <xs:simpleContent>
1634 <xs:extension base="cim:cimLong">
1635 <xs:attribute ref="cim:qualifier" use="required"/>
1636 </xs:extension>
1637 </xs:simpleContent>
1638 </xs:complexType>
1639
1640 <xs:complexType name="qualifierSArray">
1641 <xs:complexContent>
1642 <xs:extension base="cim:qualifierString"/>
1643 </xs:complexContent>
1644 </xs:complexType>
1645
1646 <!-- The following element is to be used only for defining metadata -->
1647 <xs:element name="DefaultValue" type="xs:anySimpleType" />
1648
1649 </xs:schema>

```

## 1650 A.2 Qualifiers Schema: DSP8005

1651 The following schema is an example of the qualifiers schema that is based on CIM Schema 2.13.1.  
 1652 Future versions of CIM Schema may add or delete qualifiers, which would be reflected in the  
 1653 corresponding qualifiers.xsd file.

```

1654 <?xml version="1.0" encoding="utf-8" ?>
1655 <xs:schema
1656 targetNamespace="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1657 xmlns:cimQ="http://schemas.dmtf.org/wbem/ws-cim/1/cim-schema/2/qualifiers"
1658 xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"

```

```

1659     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1660     elementFormDefault="qualified">
1661
1662     <xs:import
1663         namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1664         schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1665
1666     <xs:element name="Abstract" type="cim:qualifierBoolean"/>
1667     <xs:element name="Aggregate" type="cim:qualifierBoolean"/>
1668     <xs:element name="Aggregation" type="cim:qualifierBoolean"/>
1669     <xs:element name="ArrayType" type="cim:qualifierString"/>
1670     <xs:element name="Association" type="cim:qualifierBoolean"/>
1671     <xs:element name="BitMap" type="cim:qualifierSArray"/>
1672     <xs:element name="BitValues" type="cim:qualifierSArray"/>
1673     <xs:element name="ClassConstraint" type="cim:qualifierSArray"/>
1674     <xs:element name="Counter" type="cim:qualifierBoolean"/>
1675     <xs:element name="Composition" type="cim:qualifierBoolean"/>
1676     <xs:element name="Deprecated" type="cim:qualifierSArray"/>
1677     <xs:element name="Description" type="cim:qualifierString"/>
1678     <xs:element name="DisplayName" type="cim:qualifierString"/>
1679     <xs:element name="DN" type="cim:qualifierBoolean"/>
1680     <xs:element name="EmbeddedInstance" type="cim:qualifierBoolean"/>
1681     <xs:element name="EmbeddedObject" type="cim:qualifierBoolean"/>
1682     <xs:element name="Exception" type="cim:qualifierBoolean"/>
1683     <xs:element name="Experimental" type="cim:qualifierBoolean"/>
1684     <xs:element name="Gauge" type="cim:qualifierBoolean"/>
1685     <xs:element name="In" type="cim:qualifierBoolean"/>
1686     <xs:element name="Indication" type="cim:qualifierBoolean"/>
1687     <xs:element name="Key" type="cim:qualifierBoolean"/>
1688     <xs:element name="MappingStrings" type="cim:qualifierSArray"/>
1689     <xs:element name="Max" type="cim:qualifierUInt32"/>
1690     <xs:element name="MethodConstraint" type="cim:qualifierSArray"/>
1691     <xs:element name="Min" type="cim:qualifierUInt32"/>
1692     <xs:element name="MaxLen" type="cim:qualifierUInt32"/>
1693     <xs:element name="MaxValue" type="cim:qualifierSInt64"/>
1694     <xs:element name="MinLen" type="cim:qualifierUInt32"/>
1695     <xs:element name="MinValue" type="cim:qualifierSInt64"/>
1696     <xs:element name="Revision" type="cim:qualifierString"/>         <!-- Is Deprecated -->
1697     <xs:element name="ModelCorrespondence" type="cim:qualifierSArray"/>
1698     <xs:element name="NullValue" type="cim:qualifierString"/>
1699     <xs:element name="OctetString" type="cim:qualifierBoolean"/>
1700     <xs:element name="Out" type="cim:qualifierBoolean"/>
1701     <xs:element name="Override" type="cim:qualifierString"/>
1702     <xs:element name="Propagated" type="cim:qualifierString"/>
1703     <xs:element name="PropertyConstraint" type="cim:qualifierSArray"/>
1704     <xs:element name="Read" type="cim:qualifierBoolean"/>
1705     <xs:element name="Required" type="cim:qualifierBoolean"/>
1706     <xs:element name="Schema" type="cim:qualifierString"/>
1707     <xs:element name="Static" type="cim:qualifierBoolean"/>
1708     <xs:element name="Terminal" type="cim:qualifierBoolean"/>
1709     <xs:element name="Units" type="cim:qualifierString"/>
1710     <xs:element name="UMLPackagePath" type="cim:qualifierString"/>
1711     <xs:element name="ValueMap" type="cim:qualifierSArray"/>

```

```

1712 <xs:element name="Values" type="cim:qualifierSArray"/>
1713 <xs:element name="Version" type="cim:qualifierString"/>
1714 <xs:element name="Weak" type="cim:qualifierBoolean"/>
1715 <xs:element name="Write" type="cim:qualifierBoolean"/>
1716
1717 <!-- Qualifier defined by DMTF for a future release of CIM Schema -->
1718 <!-- Included in this version at the request of the WSDM-CIM mapping team -->
1719 <xs:element name="Correlatable" type="cim:qualifierSArray"/>
1720
1721 <!-- Following qualifiers are considered to be "Optional Qualifiers" in CIM. -->
1722 <xs:element name="Alias" type="cim:qualifierString"/>
1723 <xs:element name="Delete" type="cim:qualifierBoolean"/>
1724 <xs:element name="Expensive" type="cim:qualifierBoolean"/>
1725 <xs:element name="IfDeleted" type="cim:qualifierBoolean"/>
1726 <xs:element name="Invisible" type="cim:qualifierBoolean"/>
1727 <xs:element name="Large" type="cim:qualifierBoolean"/>
1728 <xs:element name="Provider" type="cim:qualifierString"/>
1729 <xs:element name="PropertyUsage" type="cim:qualifierString"/>
1730 <xs:element name="Syntax" type="cim:qualifierString"/>
1731 <xs:element name="SyntaxType" type="cim:qualifierString"/>
1732 <xs:element name="TriggerType" type="cim:qualifierString"/>
1733 <xs:element name="UnknownValues" type="cim:qualifierSArray"/>
1734 <xs:element name="UnsupportedValues" type="cim:qualifierSArray"/>
1735
1736 </xs:schema>

```

### 1737 A.3 Class Hierarchy Type Schema: DSP8006

1738 The complex type definition in the following schema provides the type of GEDs that describe the CIM  
 1739 Schema subclass / superclass hierarchy. The element `ClassHierarchy` may be used by protocols as  
 1740 an element in XML instance documents of a CIM instance that contains a value representing the subclass  
 1741 / superclass hierarchy of a class. Its presence as an element in an instance document would be covered  
 1742 by the `xs:any` in the WS-CIM schema of the instance's class.

```

1743 <?xml version="1.0" encoding="utf-8"?>
1744 <xs:schema
1745   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1746   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
1747   xmlns:xs="http://www.w3.org/2001/XMLSchema">
1748   <xs:complexType name="ClassHierarchyType">
1749     <xs:sequence>
1750       <xs:any minOccurs="0" namespace="##any" processContents="lax" />
1751     </xs:sequence>
1752   </xs:complexType>
1753
1754   <xs:element name="ClassHierarchy" type="ctype:ClassHierarchyType"/>
1755
1756 </xs:schema>

```



## ANNEX B (Informative)

### Conventions

1757  
1758  
1759  
1760

1761 In XML and MOF examples, an ellipsis (" . . . ") indicates omitted or optional entries that would typically  
1762 occupy the position of the ellipsis.

1763 The following conventions are followed for defining formats of entries such as URIs:

- 1764 • Literal characters within a format definition are surrounded by single quotes.
- 1765 • Names of variables within a format are in standard text and are explicitly defined by means of a  
1766 "Where: variable-name is ..." section that follows the format definition.
- 1767 • A specific value of a variable within a generalized example of a formatted entry is displayed in  
1768 *italics*.
- 1769 • Definitions of formats are case sensitive.
- 1770 • Whitespace, if any, in formats is explicitly indicated.

1771 The following typographical conventions are used:

- 1772 • `Monospace font`: CIM datatypes and element names as well as XML and WSDL element  
1773 and attribute names.
- 1774 • `Courier new 8, gray background`: Code examples

## ANNEX C (Informative)

### Examples

1775  
1776  
1777  
1778

1779 This annex contains examples of converting MOF definitions of several classes into XML Schema, WSDL  
1780 fragments, and metadata fragments. Although the classes are fictional creations used to illustrate  
1781 different features of the conversion, the classes are based on actual CIM classes.

#### 1782 C.1 MOF Definitions

1783 This clause contains the MOF definitions that are converted in these examples.

##### 1784 C.1.1 EX\_BaseComponent

```

1785 [Abstract, Version ( "2.x" ), Description (
1786     "EX_BaseComponent serves as an example base CIM class.")]
1787 class EX_BaseComponent {
1788     [Description (
1789         "A datetime value indicating when the object was installed.")]
1790     datetime InstallDate;
1791     [Description (
1792         "The Name property defines the label by which the object is "
1793         "known."),
1794         MaxLen ( 1024 ), Required]
1795     string Name;
1796     [Description (
1797         "A set of descriptive statements that can be used to describe the "
1798         "state of a Component."),
1799         ArrayType ( "Indexed" ) ]
1800     string StatusDescriptions[];
1801     [Description (
1802         "A descriptive code representing operational health of a Component."),
1803         ValueMap { "OK", "Error", "Unknown" }, MaxLen ( 10 ) ]
1804     string HealthStatus;
1805 };

```

##### 1806 C.1.2 EX\_DerivedComponent

```

1807 [Version ( "2.x" ), Description (
1808     "This class extends EX_BaseComponent.")]
1809 class EX_DerivedComponent : EX_BaseComponent {
1810     [Description (
1811         "EnabledState is an integer enumeration that indicates the "
1812         "enabled and disabled states of a derived Component."),
1813         ValueMap { "0", "1", "2", "3" },
1814         Values { "Unknown", "Other", "Enabled", "Disabled" } ]
1815     uint16 EnabledState=3;
1816     [Description (
1817         "Boolean flag indicating availability of a Component.") ]
1818     boolean AvailableFlag;
1819     [Description (
1820         "Requests that the state of the element be changed to the "
1821         "value specified in the RequestedState parameter."),
1822         ValueMap { "0", "1", "2", "3..32767", "32768..65535" },
1823         Values { "Completed with No Error", "Not Supported",

```

```

1824         "Failed", "DMTF Reserved", "Vendor Specific" } ]
1825     uint32 RequestStateChange(
1826         [IN, Description (
1827             "The state requested for the Component."),
1828             ValueMap { "2", "3", "4" },
1829             Values { "Enabled", "Disabled" "Shutdown" } ]
1830     uint16 RequestedState,
1831     [IN ( false ), OUT, Description (
1832         "Reference to an instance of some class (undefined in this "
1833         "example) that is returned upon completion of the operation.")]
1834     CIM_SomeClass REF ResultClass,
1835     [IN, Description (
1836         "A timeout period that specifies the maximum amount of "
1837         "time that the client expects the transition to the new "
1838         "state to take. ")]
1839     datetime TimeoutPeriod);
1840 };

```

### 1841 C.1.3 EX\_AssociationComponent

```

1842     [Association, Version ( "2.x" ), Description (
1843         "Indicates that two entites are associated.")]
1844 class EX_Association {
1845     [Key, Description (
1846         "AssociatingComponent represents one Component is "
1847         "associated with the Component referenced as AssociatedComponent.")]
1848     EX_BaseComponent REF AssociatingComponent;
1849     [Key, Description (
1850         "AssociatedComponent represents another Component (up to 4) that "
1851         "is associated with the Component referenced as "
1852         "AssociatingComponent."),
1853         Max ( 4 )]
1854     EX_BaseComponent REF AssociatedComponent;
1855     [Description (
1856         "The point in time that the Components were associated.")]
1857     datetime WhenAssociated;
1858     [Description (
1859         "Boolean indicating whether the association is maintained.")]
1860     boolean AssocMaintained;
1861 };

```

## 1862 C.2 XSD

1863 This clause shows the XML Schema files that would result from the application of this specification to the  
 1864 preceding example CIM classes.

### 1865 C.2.1 EX\_BaseComponent

```

1866 <?xml version="1.0" encoding="utf-8"?>
1867 <xs:schema
1868     targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1869     xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_BaseComponent"
1870     xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1871     xmlns:xs="http://www.w3.org/2001/XMLSchema"
1872     ...>
1873 <xs:import
1874     namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1875     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1876 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>

```

```

1877 <xs:element name="Name">
1878   <xs:complexType>
1879     <xs:simpleContent>
1880       <xs:restriction base="cim:cimString">
1881         <xs:maxLength value="1024"/>
1882         <xs:anyAttribute namespace="##any" processContents="lax"/>
1883       </xs:restriction>
1884     </xs:simpleContent>
1885   </xs:complexType>
1886 </xs:element>
1887 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1888 <xs:element name="HealthStatus" nillable="true">
1889   <xs:complexType>
1890     <xs:simpleContent>
1891       <xs:restriction base="cim:cimString">.
1892         <xs:enumeration value="OK"/>
1893         <xs:enumeration value="Error"/>
1894         <xs:enumeration value="Unknown"/>
1895         <xs:maxLength value="10"/>
1896         <xs:anyAttribute namespace="##any" processContents="lax"/>
1897       </xs:restriction>
1898     </xs:simpleContent>
1899   </xs:complexType>
1900 </xs:element>
1901 <xs:complexType name="EX_BaseComponent_Type">
1902   <xs:sequence>
1903     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1904     <xs:element ref="class:InstallDate" minOccurs="0"/>
1905     <xs:element ref="class:Name"/>
1906     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1907     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1908   </xs:sequence>
1909   <xs:anyAttribute namespace="##any" processContent="lax"/>
1910 </xs:complexType>
1911 <xs:element name="EX_BaseComponent" type="class:EX_BaseComponent_Type"/>
1912 </xs:schema>

```

### 1913 C.2.2 EX\_DerivedComponent

```

1914 <?xml version="1.0" encoding="utf-8"?>
1915 <xs:schema
1916   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1917   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_DerivedComponent"
1918   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
1919   xmlns:xs="http://www.w3.org/2001/XMLSchema"
1920   ...>
1921 <xs:import
1922   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
1923   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
1924 <xs:element name="InstallDate" type="cim:cimDateTime" nillable="true"/>
1925 <xs:element name="Name">
1926   <xs:complexType>
1927     <xs:simpleContent>
1928       <xs:restriction base="cim:cimString">
1929         <xs:maxLength value="1024"/>
1930         <xs:anyAttribute namespace="##any" processContents="lax"/>
1931       </xs:restriction>
1932     </xs:simpleContent>
1933   <xs:complexType>

```

```

1934 </xs:element>
1935 <xs:element name="StatusDescriptions" type="cim:cimString"/>
1936 <xs:element name="HealthStatus" nillable="true">
1937   <xs:complexType>
1938     <xs:simpleContent>
1939       <xs:restriction base="cim:cimString">
1940         <xs:enumeration value="OK"/>
1941         <xs:enumeration value="Error"/>
1942         <xs:enumeration value="Unknown"/>
1943         <xs:maxLength value="10"/>
1944         <xs:anyAttribute namespace="##any" processContents="lax"/>
1945       </xs:restriction>
1946     </xs:simpleContent>
1947   </xs:complexType>
1948 </xs:element>
1949 <xs:element name="EnabledState" nillable="true">
1950   <xs:complexType>
1951     <xs:simpleContent>
1952       <xs:restriction base="cim:cimUnsignedShort">
1953         <xs:enumeration value="0"/>
1954         <xs:enumeration value="1"/>
1955         <xs:enumeration value="2"/>
1956         <xs:enumeration value="3"/>
1957         <xs:anyAttribute namespace="##any" processContents="lax"/>
1958       </xs:restriction>
1959     </xs:simpleContent>
1960   </xs:complexType>
1961 </xs:element>
1962 <xs:element name="AvailableFlag" type="cim:cimBoolean" nillable="true"/>
1963 <xs:complexType name="EX_DerivedComponent_Type">
1964   <xs:sequence>
1965     <xs:element ref="class:AvailableFlag" minOccurs="0"/>
1966     <xs:element ref="class:EnabledState" minOccurs="0"/>
1967     <xs:element ref="class:HealthStatus" minOccurs="0"/>
1968     <xs:element ref="class:InstallDate" minOccurs="0"/>
1969     <xs:element ref="class:Name"/>
1970     <xs:element ref="class:StatusDescriptions" minOccurs="0" maxOccurs="unbounded"/>
1971     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
1972   </xs:sequence>
1973   <xs:anyAttribute namespace="##any" processContent="lax"/>
1974 </xs:complexType>
1975 <xs:element name="EX_DerivedComponent" type="class:EX_DerivedComponent_Type"/>
1976 <xs:element name="RequestStateChange_INPUT">
1977   <xs:complexType>
1978     <xs:sequence>
1979       <xs:element name="RequestedState" nillable="true">
1980         <xs:complexType>
1981           <xs:simpleContent>
1982             <xs:restriction base="cim:cimUnsignedShort">
1983               <xs:enumeration value="2"/>
1984               <xs:enumeration value="3"/>
1985               <xs:enumeration value="4">
1986                 <xs:anyAttribute namespace="##any" processContents="lax"/>
1987               </xs:restriction>
1988             </xs:simpleContent>
1989           </xs:complexType>
1990         </xs:element>
1991       <xs:element name="TimeoutPeriod" type="cim:cimDateTime" nillable="true"/>

```

```

1992     </xs:sequence>
1993   </xs:complexType>
1994 </xs:element>
1995 <xs:element name="RequestStateChange_OUTPUT">
1996   <xs:complexType>
1997     <xs:sequence>
1998       <xs:element name="ResultClass" type="cim:cimReference" nillable="true"/>
1999       <xs:element name="ReturnValue" nillable="true">
2000     <xs:complexType>
2001       <xs:simpleContent>
2002         <xs:restriction base="cim:cimAnySimpleType">
2003           <xs:simpleType>
2004             <xs:union>
2005               <xs:simpleType>
2006                 <xs:restriction base="xs:unsignedInt">
2007                   <xs:enumeration value="0"/>
2008                   <xs:enumeration value="1"/>
2009                   <xs:enumeration value="2"/>
2010                 </xs:restriction>
2011               </xs:simpleType>
2012               <xs:simpleType>
2013                 <xs:restriction base="xs:unsignedInt">
2014                   <xs:minInclusive value="3"/>
2015                   <xs:maxInclusive value="32767"/>
2016                 </xs:restriction>
2017               </xs:simpleType>
2018               <xs:simpleType>
2019                 <xs:restriction base="xs:unsignedInt">
2020                   <xs:minInclusive value="32768"/>
2021                   <xs:maxInclusive value="65535"/>
2022                 </xs:restriction>
2023               </xs:simpleType>
2024             </xs:union>
2025           </xs:simpleType>
2026           <xs:anyAttribute namespace="##any" processContents="lax"/>
2027         </xs:restriction>
2028       </xs:simpleContent>
2029     </xs:complexType>
2030   </xs:element>
2031 </xs:sequence>
2032 </xs:complexType>
2033 </xs:element>
2034 </xs:schema>

```

### 2035 C.2.3 EX\_AssociationComponent

```

2036 <?xml version="1.0" encoding="utf-8"?>
2037 <xs:schema
2038   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2039   xmlns:cim="http://schemas.dmtf.org/wbem/wscim/1/common"
2040   xmlns:class="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/EX_AssociationComponent"
2041   xmlns:xs="http://www.w3.org/2001/XMLSchema"
2042   ...>
2043 <xs:import
2044   namespace="http://schemas.dmtf.org/wbem/wscim/1/common"
2045   schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/common.xsd"/>
2046 <xs:element name="AssociatingComponent" type="cim:cimReference"/>
2047 <xs:element name="AssociatedComponent" type="cim:cimReference"/>
2048 <xs:element name="WhenAssociated" type="cim:cimDateTime" nillable="true"/>

```

```

2049 <xs:element name="AssocMaintained" type="cim:cimBoolean" nillable="true"/>
2050 <xs:complexType name="EX_AssociationComponent_Type">
2051   <xs:sequence>
2052     <xs:element ref="class:AssociatedComponent" />
2053     <xs:element ref="class:AssociatingComponent" />
2054     <xs:element ref="class:AssocMaintained" minOccurs="0"/>
2055     <xs:element ref="class:WhenAssociated" minOccurs="0"/>
2056     <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2057   </xs:sequence>
2058   <xs:anyAttribute namespace="##any" processContent="lax"/>
2059 </xs:complexType>
2060 <xs:element name="EX_AssociationComponent" type="class:EX_AssociationComponent_Type"/>
2061 </xs:schema>

```

## 2062 C.2.4 Class Hierarchy Schema

```

2063 <?xml version="1.0" encoding="utf-8"?>
2064 <xs:schema
2065   targetNamespace="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2066   xmlns:chier="http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/classhierarchy"
2067   xmlns:ctype="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2068   xmlns:xs="http://www.w3.org/2001/XMLSchema">
2069   <xs:import
2070     namespace="http://schemas.dmtf.org/wbem/wscim/1/classhiertype"
2071     schemaLocation="http://schemas.dmtf.org/wbem/wscim/1/classhiertype.xsd"/>
2072   <xs:element name="EX_BaseComponent_Class">
2073     <xs:complexType>
2074       <xs:complexContent>
2075         <xs:restriction base="ctype:ClassHierarchyType" />
2076       </xs:complexContent>
2077     </xs:complexType>
2078   </xs:element>
2079   <xs:element name="EX_DerivedComponent_Class">
2080     <xs:complexType>
2081       <xs:complexContent>
2082         <xs:restriction base="ctype:ClassHierarchyType">
2083           <xs:sequence>
2084             <xs:element ref="chier:EX_BaseComponent_Class" />
2085           </xs:sequence>
2086         </xs:restriction>
2087       </xs:complexContent>
2088     </xs:complexType>
2089   </xs:element>
2090   <xs:element name="EX_AssociationComponent_Class">
2091     <xs:complexType>
2092       <xs:complexContent>
2093         <xs:restriction base="ctype:ClassHierarchyType" />
2094       </xs:complexContent>
2095     </xs:complexType>
2096   </xs:element>
2097 </xs:schema>

```

## 2098 C.3 WSDL Fragments

2099 This clause contains the WSDL fragments (`wsdl:message`, `wsdl:operation`) that would result from  
2100 the application of this specification to the `EX_DerivedComponent` class. This class specifies only one  
2101 method, `RequestStateChange`.

```

2102 <?xml version="1.0" encoding="utf-8"?>

```

```

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

```

## 2131 C.4 MetaData Fragments

2132 Metadata fragments are generated from the qualifiers that are associated with a class, property,  
 2133 reference, method, or parameter. XML documents that incorporate these fragments must import the cim  
 2134 and cimQ namespaces.

### 2135 C.4.1 EX\_BaseComponent

#### 2136 C.4.1.1 Class Qualifiers

```

2137 <cimQ:Abstract cim:qualifier="true">true</cimQ:Abstract>
2138 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2139 <cimQ:Description cim:qualifier="true">
2140     EX_BaseComponent serves as an example base CIM class.
2141 </cimQ:Description>

```

#### 2142 C.4.1.2 Property Qualifiers

##### 2143 C.4.1.2.1 HealthStatus

```

2144 <cimQ:Description cim:qualifier="true">
2145     A descriptive code of the operational health of a Component.
2146 </cimQ:Description>
2147 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
2148 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>

```

##### 2149 C.4.1.2.2 InstallDate

```

2150 <cimQ:Description cim:qualifier="true">
2151     EX_BaseComponent serves as an example base CIM class.
2152 </cimQ:Description>
2153 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>

```



**2154 C.4.1.2.3 Name**

```

2155 <cimQ:Description cim:qualifier="true">
2156     The Name property defines the label by which the object is known.
2157 </cimQ:Description>
2158 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
2159 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

```

**2160 C.4.1.2.4 StatusDescriptions**

```

2161 <cimQ:Description cim:qualifier="true">
2162     A set of descriptive statements that can be used to describe the state of an Component.
2163 </cimQ:Description>
2164 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

```

**2165 C.4.2 EX\_DerivedComponent****2166 C.4.2.1 Class Qualifiers**

```

2167 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2168 <cimQ:Description cim:qualifier="true">
2169     This class extends EX_BaseComponent.
2170 </cimQ:Description>

```

**2171 C.4.2.2 Property Qualifiers****2172 C.4.2.2.1 AvailableFlag**

```

2173 <cimQ:Description cim:qualifier="true">
2174     Boolean flag indicating availability of a Component.

```

**2175 C.4.2.2.2 EnabledState**

```

2176 <cimQ:Description cim:qualifier="true">
2177     EnabledState is an integer enumeration that indicates the enabled
2178     and disabled states of a derived Component.
2179 </cimQ:Description>
2180 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
2181 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
2182 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
2183 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
2184 <cimQ:Values cim:qualifier="true">Unknown</cimQ:Values>
2185 <cimQ:Values cim:qualifier="true">Other</cimQ:Values>
2186 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
2187 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
2188 <cim:DefaultValue xsi:type="xs:uint16">3</cim:DefaultValue>
2189 HealthStatus
2190 <cimQ:Description cim:qualifier="true">
2191     A descriptive code of the operational health of a Component.
2192 </cimQ:Description>
2193 <cimQ:ValueMap cim:qualifier="true">OK</cimQ:ValueMap>
2194 <cimQ:ValueMap cim:qualifier="true">Error</cimQ:ValueMap>
2195 <cimQ:ValueMap cim:qualifier="true">Unknown</cimQ:ValueMap>
2196 </cimQ:Description>

```

**2197 C.4.2.2.3 InstallDate**

```

2198 <cimQ:Description cim:qualifier="true">
2199     EX_BaseComponent serves as an example base CIM class.
2200 </cimQ:Description>

```

**2201 C.4.2.2.4 Name**

```

2202 <cimQ:Description cim:qualifier="true">
2203     The Name property defines the label by which the object is known.
2204 </cimQ:Description>
2205 <cimQ:MaxLen cim:qualifier="true">1024</cimQ:MaxLen>
2206 <cimQ:Required cim:qualifier="true">true</cimQ:Required>

```

**2207 C.4.2.2.5 StatusDescriptions**

```

2208 <cimQ:Description cim:qualifier="true">
2209     A set of descriptive statements that can used to describe the state of an Component.
2210 </cimQ:Description>
2211 <cimQ:ArrayType cim:qualifier="true">Indexed</cimQ:ArrayType>

```

**2212 C.4.2.2.6 AvailableFlag**

```

2213 <cimQ:Description cim:qualifier="true">
2214     Boolean flag indicating availability of a Component.
2215 </cimQ:Description>

```

**2216 C.4.2.3 Method and Parameter Qualifiers****2217 C.4.2.3.1 RequestStatusChange Method**

```

2218 <cimQ:Description cim:qualifier="true">
2219     Requests that the state of the element be changed to the value
2220     specified in the RequestedState parameter.
2221 </cimQ:Description>
2222 <cimQ:ValueMap cim:qualifier="true">0</cimQ:ValueMap>
2223 <cimQ:ValueMap cim:qualifier="true">1</cimQ:ValueMap>
2224 <cimQ:ValueMap cim:qualifier="true">..</cimQ:ValueMap>
2225 <cimQ:ValueMap cim:qualifier="true">4096</cimQ:ValueMap>
2226 <cimQ:ValueMap cim:qualifier="true">4100..32767</cimQ:ValueMap>
2227 <cimQ:ValueMap cim:qualifier="true">32768..65535</cimQ:ValueMap>
2228 <cimQ:Values cim:qualifier="true">Completed with No Error</cimQ:Values>
2229 <cimQ:Values cim:qualifier="true">Not Supported</cimQ:Values>
2230 <cimQ:Values cim:qualifier="true">Unknown or Unspecified Error</cimQ:Values>
2231 <cimQ:Values cim:qualifier="true">Failed</cimQ:Values>
2232 <cimQ:Values cim:qualifier="true">DMTF Reserved</cimQ:Values>
2233 <cimQ:Values cim:qualifier="true">Vendor Specific</cimQ:Values>

```

**2234 C.4.2.3.2 RequestedState Parameter**

```

2235 <cimQ:Description cim:qualifier="true">
2236     The state requested for the Component.
2237 </cimQ:Description>
2238 <cimQ:In cim:qualifier="true">true</cimQ:In>
2239 <cimQ:ValueMap cim:qualifier="true">2</cimQ:ValueMap>
2240 <cimQ:ValueMap cim:qualifier="true">3</cimQ:ValueMap>
2241 <cimQ:ValueMap cim:qualifier="true">4</cimQ:ValueMap>
2242 <cimQ:Values cim:qualifier="true">Enabled</cimQ:Values>
2243 <cimQ:Values cim:qualifier="true">Disabled</cimQ:Values>
2244 <cimQ:Values cim:qualifier="true">Shutdown</cimQ:Values>

```

**2245 C.4.2.3.3 ResultClass Parameter**

```

2246 <cimQ:Description cim:qualifier="true">
2247     Reference to an instance of some class (undefined in this example)
2248     that is returned upon completion of the operation.

```

```

2249 </cimQ:Description>
2250 <cimQ:Out cim:qualifier="true">true</cimQ:Out>
2251 <cimQ:In cim:qualifier="true">false</cimQ:In>

```

#### 2252 **C.4.2.3.4 TimeoutPeriod Parameter**

```

2253 <cimQ:Description cim:qualifier="true">
2254     A timeout period that specifies the maximum amount of time that the
2255     client expects the transition to the new state to take.
2256 </cimQ:Description>
2257 <cimQ:In cim:qualifier="true">true</cimQ:In>

```

### 2258 **C.4.3 EX\_AssociationComponent**

#### 2259 **C.4.3.1 Class Qualifiers**

```

2260 <cimQ:Version cim:qualifier="true">2.x</cimQ:Version>
2261 <cimQ:Description cim:qualifier="true">
2262     Indicates that two entites are associated.
2263 </cimQ:Description>
2264 <cimQ:Association cim:qualifier="true">true</cimQ:Association>

```

#### 2265 **C.4.3.2 Property Qualifiers**

##### 2266 **C.4.3.2.1 AssociatedComponent**

```

2267 <cimQ:Key cim:qualifier="true">true</cimQ:Key>
2268 <cimQ:Description cim:qualifier="true">
2269     AssociatedComponent represents another Component (up to 4) that is associated
2270     with the Component referenced as AssociatingComponent.
2271 </cimQ:Description>
2272 <cimQ:Max cim:qualifier="true">4</cimQ:Max>

```

##### 2273 **C.4.3.2.2 AssociatingComponent**

```

2274 <cimQ:Key cim:qualifier="true">true</cimQ:Key>
2275 <cimQ:Description cim:qualifier="true">
2276     An AssociatingComponent represents one Component is associated with the
2277     component referenced as AssociatedComponent.
2278 </cimQ:Description>

```

##### 2279 **C.4.3.2.3 AssocMaintained**

```

2280 <cimQ:Description cim:qualifier="true">
2281     Boolean indicating whether the association is maintained.
2282 </cimQ:Description>

```

##### 2283 **C.4.3.2.4 WhenAssociated**

```

2284 <cimQ:Description cim:qualifier="true">
2285     The point in time that the Components were associated.
2286 </cimQ:Description>

```

## ANNEX D (Informative)

### Collation Optimization Available to Implementers

#### 2291 D.1 Sorting with Limited Character Set

2292 The character set permitted in CIM identifiers is limited to

2293 - U+0030..U+0039 (digits 0-9)

2294 - U+0041..U+005A (alphabetic A-Z)

2295 - U+0061..U+007A (alphabetic a-z)

2296 - U+005F (underscore)

2297 - U+0080..U+FFEF (the rest of Unicode)

2298 The intention of the preferred collation is that the property name identifier strings should be ordered by a  
2299 binary sorting of big-endian UCS-2 representation of the characters. For example, ABC, ABc, and AbC  
2300 sort in this order because

2301 ABC 00 41 00 42 00 43

2302 ABc 00 41 00 42 00 63

2303 AbC 00 41 00 62 00 43

2304 (and accented As sort in order by Unicode number)

2305 If the identifiers of a MOF do not use the full range of characters, optimization can be applied to the  
2306 sorting of property identifiers. Specifically, a simple approach exists for the commonly observed case  
2307 where MOF characters are limited to ASCII-7, i.e., character values < 0x7F.

2308 Most or all MOFs that you will encounter contain only (a subset of) ASCII-7 characters, that is, characters  
2309 in the range 0x00 to 0x7F. For such MOFs, it is not necessary to cast the property identifier strings into  
2310 Unicode representation at all. If all the characters are ASCII-7, then the strings can be sorted as simple  
2311 one-byte sequences.

2312 If the internal representation of character strings (in a CIMOM, protocol adapter, or client application, etc.)  
2313 is UTF-8, note that ASCII-7 characters are represented unmodified in UTF-8.

2314 Suggested algorithm: Pre-scan the MOF, or at least the set of property identifiers, for characters > 0x7F.  
2315 If the set contains no such characters > 0x7F, then sort the strings as simple one-byte octet strings.

2316 This case can be used for all currently published DMTF MOFs.

#### 2317 D.2 Note of Caution Concerning Collation

2318 The default Unicode Collation Algorithm orders properties differently from the current DMTF practice. All  
2319 published CIM XML schemata order properties as described here (sorting by Unicode value). The UCA,  
2320 by default, uses a different ordering even for the subset of ASCII-7 characters: lower case characters sort  
2321 before upper case characters. This results in differences in some cases, where, for example, "CPU" and  
2322 "Caption" appear in one order in the published XSD files but sort in the other order using the UCA.

2323 The intention of the preferred algorithm is to retain current DMTF practice. Properties in XSDs will  
2324 continue to be in upper-before-lower order. Some existing WS-Man service implementations may be  
2325 using the default Unicode Collation Algorithm. These implementations will have to become compatible  
2326 with existing practice.