1

2 **Document Number: DSP0201**

3 **Date: 2014-01-16**

4 **Version: 2.4.0**

5 # Representation of CIM in XML

6 **Document Type: Specification**

7 **Document Status: DMTF Standard**

8 **Document Language: en-US**

9 | Copyright Notice

10 | Copyright © 1999-2014 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

31

CONTENTS

135 # Tables

138

139   Foreword

140   The *Representation of CIM in XML* (DSP0201) was prepared by the DMTF CIM-XML Working Group.

141   DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
142   management and interoperability.

143   Introduction

144   This document defines an XML grammar, written in document type definition (DTD), which can be used to
145   represent both Common Information Model (CIM) declarations (classes, instances and qualifiers) and
146   CIM-XML messages for use by DSP0200 (*CIM Operations over HTTP*).

147   For convenience, the complete unannotated DTD is available as a separate document (DSP0203).

148   The same XML grammar is also described using XSD as DSP8044.

149   CIM information could be represented within XML in many different ways. In the interest of interoperability
150   between different implementations of CIM, there is an obvious requirement for standardization of this
151   representation. The following criteria have been applied in the design of the representation presented
152   here:

153   •   Fully standardized technologies are used wherever possible, in preference to Working Drafts.
154   Where use is made of a Working Draft, the intention is to track the changes to the Working Draft
155   in this specification.
156   •   Completeness is favored over conciseness (all aspects of CIM should be modeled).

157   Although this document makes no restrictions on the use of this mapping, a number of possible usage
158   scenarios exist for which the mapping should provide:

159   •   XML documents conforming to this mapping that express CIM-XML declarations should be
160   capable of being rendered or transformed using standard techniques into other formats. In
161   particular, the mapping should contain sufficient information to be rendered into Managed
162   Object Format (MOF) syntax (DSP0004).
163   •   The mapping should be applicable to the wire-level representation of CIM-XML messages
164   defined by DSP0200.

A Note on Rendering to MOF

166   The subset of the DTD for CIM presented in this specification that concerns object declarations (identified
167   by the element DECLARATION) is intended to allow expression of CIM objects in XML sufficient for
168   rendering into a number of formats, including MOF.

169   The semantic content of a MOF file is fully captured by the DTD presented herein, which makes it
170   possible to express any MOF conformant to DSP0004 in an equivalent XML representation using this
171   DTD. This includes the ability to express any of the standard MOF pragmas defined in DSP0004, with the
172   exception of the `locale` and `instancelocale` pragmas (which are subjects for further study in the
173   context of localization support within CIM).

174   Note that the Processing Instruction mechanism defined by XML is the means by which bespoke
175   pragmas may be added to an XML document in an analogous manner to the #pragma extension
176   mechanism defined for MOF. The format of such PIs is necessarily outside the scope of this document.

A Note on Mapping Choices

178   There are two fundamentally different models for mapping CIM in XML:

179   •   A *Schema Mapping* is one in which the XML schema is used to describe the CIM classes, and
180   CIM Instances are mapped to valid XML documents for that schema. (Essentially this means
181   that each CIM class generates its own DTD fragment, the XML element names of which are
182   taken directly from the corresponding CIM element names.)
183   •   A *Metaschema Mapping* is one in which the XML schema is used to describe the CIM
184   metaschema, and both CIM classes and instances are valid XML documents for that schema.
185   (In other words, the DTD is used to describe in a generic fashion the notion of a CIM class or

186 instance. CIM element names are mapped to XML attribute or element values rather than XML
187 element names.)

188 Although employing a schema mapping has obvious benefits (more validation power and a slightly more
189 intuitive representation of CIM in XML), the metaschema mapping is adopted here for the following
190 reasons:

191 • It requires only one standardized metaschema DTD for CIM rather than an unbounded number
192 of DTDs. This considerably reduces the complexity of management and administration of XML
193 mappings.
194 • An XML DTD does not allow an unordered list of elements. In a static mapping, this restriction
195 would require one of the following actions:

196 – Fixing an arbitrary order for property, method, and qualifier lists (making it harder for a
197 receiving application to process)
198 – Defining a very unwieldy mapping that accounts for all list orderings explicitly (and whose
199 size would grow exponentially with the number of list elements)

200 • In a schema mapping, the names of CIM schema elements (class, property, qualifier, and
201 method names) populate the XML element namespace. To replicate the scoping rules on CIM
202 element names within an XML DTD, it would be necessary to employ XML namespaces to
203 define XML schema to a per-property level of granularity. This would be extremely cumbersome
204 to administer and process. A metaschema mapping introduces only a small, fixed number of
205 terms into the XML element namespace (such as Class, Instance, Property, and so on). As an
206 alternative to the introduction of additional XML namespaces, some renaming of CIM elements
207 could be used (for example, prefixing a qualifier name with the name of its owning property and
208 its owning class), but this would result in XML documents that are verbose and difficult to
209 understand.
210 • Although a schema mapping could allow XML-based validation of instances against classes,
211 this would be possible only if the entire class hierarchy were flattened prior to mapping the CIM
212 class to an XML schema. If this flattening was not performed, inherited properties might be
213 absent from the DTD, which would cause validation to fail against an instance that included the
214 value of an inherited property.

215

216   # Representation of CIM in XML

217 ## 1   Scope

218 The [Extensible Markup Language](#) (XML) is a simplified subset of SGML that offers powerful and
219 extensible data modeling capabilities. An *XML document* is a collection of data represented in XML. An
220 *XML schema* is a grammar that describes the format of an XML document. An XML document is
221 described as *valid* if it has an associated XML schema to which it conforms.

222 The [Common Information Model](#) (CIM) is an object-oriented information model defined by the DMTF that
223 provides a conceptual framework for describing management data.

224 This document defines a standard for the representation of CIM elements and messages in XML.

225 ## 2   Normative References

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

229 ANSI/IEEE 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, August 1985,
230 http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=30711

231 DMTF DSP0004, *Common Information Model (CIM) Infrastructure 2.7*,
232 http://www.dmtf.org/standards/published_documents/DSP0004_2.7.pdf

233 DMTF DSP0200, *CIM Operations over HTTP 1.4*,
234 http://www.dmtf.org/standards/published_documents/DSP0200_1.4.pdf

235 IETF RFC1034 *Domain Names - Concepts and Facilities*, November 1987,
236 http://tools.ietf.org/html/rfc1034

237 IETF RFC3986, Uniform Resource Identifiers (URI): Generic Syntax, August 1998 344
238 http://tools.ietf.org/html/rfc2396

239 ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*,
240 http://standards.iso.org/ittf/PubliclyAvailableStandards/c039921_ISO_IEC_10646_2003(E).zip

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

243 *W3C Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, September 2006,
244 http://www.w3.org/TR/2006/REC-xml-20060816/

245 *W3C Namespaces in XML 1.0 (Second Edition)*, W3C Recommendation, August 2006,
246 http://www.w3.org/TR/REC-xml-names/

247 ## 3   Terms and Definitions

248 In this document, some terms have a specific meaning beyond the normal English meaning. Those terms
249 are defined in this clause.

250 The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"),
251 "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described

252 in ISO/IEC Directives, Part 2, Annex H. The terms in parenthesis are alternatives for the preceding term,
253 for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that
254 ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional
255 alternatives shall be interpreted in their normal English meaning.

256 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
257 described in ISO/IEC Directives, Part 2, Clause 5.

258 The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC
259 Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do
260 not contain normative content. Notes and examples are always informative elements.

261 The terms defined in DSP0004 and DSP0200 apply to this document. The following additional terms are
262 used in this document. Some additional more detailed terms are defined throughout the subclauses of
263 this document.

264 **3.1**

265 **CIM element**

266 one of the following components of the CIM metamodel used to define a schema: Class, instance,
267 property, method, parameter, or qualifier

268 **3.2**

269 **CIM object**

270 a namespace, class, instance, or qualifier that is defined in a CIM-XML declaration or accessible in a
271 WBEM server.

272 **3.3**

273 **CIM-XML declaration**

274 a declaration of CIM objects (classes, instances and qualifiers), using the DECLARATION element
275 defined in this specification. Note that "CIM declaration" was used for this term before version 2.4 of this
276 specification.

277 **3.4**

278 **CIM-XML message**

279 a request or response message in the CIM-XML protocol, using the MESSAGE element defined in this
280 specification. Note that "CIM message" was used for this term before version 2.4 of this specification.

281 **3.5**

282 **CIM-XML consumer**

283 a WBEM server, client or listener that receives a CIM-XML message, or a program that consumes a CIM-
284 XML declaration

285 **3.6**

286 **CIM-XML producer**

287 a WBEM server, client or listener that sends a CIM-XML message, or a program that produces a CIM-
288 XML declaration

289 **3.7**

290 **CIM-XML protocol**

291 the WBEM protocol that uses the CIM operations over HTTP defined in DSP0200 and the representation
292 of CIM in XML defined in this specification

293   **3.8**

294   **CIM-XML schema**

295   the XML schema for representing CIM in XML, defined by this specification

296   **3.9**

297   **numeric character reference**

298   an escaped UCS character, as defined in clause 4.1 of the W3C XML specification as *character*
299   *reference*, using the hexadecimal or decimal representation of its UCS code point. Examples are
300   "``" for the carriage return character or "`A`" for the character "`A`".

301   **3.10**

302   **UCS character**

303   A character from the Universal Multiple-Octet Coded Character Set (UCS) defined in ISO/IEC 10646. Also
304   known as Unicode character. For an overview on UCS characters in CIM, see DSP0004.

305   **3.11**

306   **whitespace**

307   one or more consecutive occurrences of any of the characters space (U+0020), carriage return
308   (U+000D), line feed (U+000A) or horizontal tab (U+0009), consistent with the whitespace definition in the
309   W3C XML specification.

310   **3.12**

311   **XML element**

312   a component of XML that is defined using the ELEMENT construct in the DTD

313   4   Symbols and Abbreviated Terms

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

315   **4.1**
316   **CIM**

317   Common Information Model, defined in DSP0004

318   **4.2**
319   **DTD**

320   Document Type Definition, defined in the W3C XML specification

321   **4.3**
322   **MOF**

323   Managed Object Format, defined in DSP0004

324   **4.4**
325   **XML**

326   Extensible Markup Language, defined in the W3C XML specification

327   **4.5**
328   **XSD**

329   XML Schema Definition, defined in the W3C XSD specifications

330 # 5 CIM-XML Schema Reference

331 This clause describes the CIM-XML schema using DTD. DSP0203 defines the same DTD as this
332 specification without any annotations. DSP8044 defines the CIM-XML schema using XSD.

333 In case of differences between these three documents, this specification (DSP0201) overrules the other
334 two.

335 ## 5.1 General

336 ### 5.1.1 Escaping, Whitespace Handling, Character Repertoire

337 This clause defines the rules for escaping of CIM values in CIM-XML.

338 This clause uses the term "CIM value" to refer to values at the CIM level (e.g. the value of a property of a
339 CIM instance). Values at the level of XML elements and attributes are referred to as "content of the
340 element" or "attribute value", consistent with the W3C XML specification (for example, the content of a
341 VALUE element may represent the value of a property of a CIM instance).

342 #### 5.1.1.1 XML Clarifications and Amendments

343 The rules for escaping and handling of whitespace defined in the W3C XML specification apply to this
344 specification, with the following clarifications and amendments:

345 • The W3C XML specification uses the terms "XML processor" and "application". CIM-XML does
346 not distinguish software layers within a WBEM client, server or listener.

347 In this specification, any definitions in the W3C XML specification that use these two terms shall
348 be interpreted as if both of these software layers were within the WBEM client, server or
349 listener.

350 • The W3C XML specification defines in clause 2.10 that any whitespace not in XML markup shall
351 be passed by the XML processor to the application. Further, it defines in clause 2.10 that all
352 characters shall be preserved. In clause 2.11, it defines that carriage return characters
353 (U+000D) with and without directly following line feed characters (U+000A) shall be converted
354 into line feed characters when processing external parsed entities (that is, XML files).

355 This specification defines certain elements and attributes or certain uses thereof as character-
356 preserving or whitespace-tolerant. Clauses 5.1.1.2 and 5.1.1.3 define the escaping and
357 whitespace handling rules for character-preserving and whitespace-tolerant elements and
358 attributes.

359 • The W3C XML specification defines in clause 2.4 provisions for escaping. It does not explicitly
360 state how those provisions apply in the context of a protocol that uses XML in its payload., and
361 it does not explicitly state whether any character may be escaped using numeric character
362 references (vs. just certain special characters).

363 This specification defines rules on escaping in 5.1.1.4.

364 • The W3C XML specification defines in clause 2.7 that character data may be represented using
365 CDATA sections (`<![CDATA[ ... ]]>`), but it is not clear from the text whether attribute
366 values are considered character data for this purpose. The syntax in its BNF rule [10] clarifies
367 that attribute values cannot be represented using CDATA sections.

368 This specification points that out in 5.1.1.4.3.

369   **5.1.1.2   Character-preserving Elements and Attributes**

370   This specification defines certain XML elements and attributes or uses thereof to be character-preserving.

371   For character-preserving elements, attributes, or uses thereof, the following applies:

372   •   CIM-XML producers shall set the content of the element or the value of the attribute from the
373   CIM value with only the following transformation:

374   1)   Escaping as defined in 5.1.1.4.

375   •   CIM-XML consumers shall set the CIM value from the content of the element or the value of the
376   attribute with only the following transformation:

377   1)   Unescaping as defined in 5.1.1.4.

378   NOTE: For character-preserving elements and attributes, all characters are preserved, including any kind of
379   whitespace characters and specifically including any carriage return characters (U+000D), regardless of their position
380   within a string (that is, even if they are followed by line feed, U+000A).

381   **5.1.1.3   Whitespace-tolerant Elements and Attributes**

382   Any XML elements, attributes, or uses thereof not explicitly defined to be character-preserving (see
383   5.1.1.2) are considered whitespace-tolerant.

384   For whitespace-tolerant elements, attributes, or uses thereof, the following applies:

385   •   CIM-XML producers shall set the content of the element or the value of the attribute from the
386   CIM value with only the following transformations, in the stated order:

---

387   **DEPRECATED**

388   1)   Adding zero or more leading or trailing whitespace characters of the following set of
389   characters: U+0009 (horizontal tab), U+000A (line feed), U+0020 (space). Adding
390   such leading or trailing characters is discouraged and is described only for
391   compatibility with implementations of earlier versions of this specification.

392   **DEPRECATED**

---

393   2)   Escaping as defined in 5.1.1.4.

394   •   CIM-XML consumers shall set the CIM value from the content of the element or the value of the
395   attribute with only the following transformations, in the stated order:

396   1)   Unescaping as defined in 5.1.1.4.

---

397   **DEPRECATED**

398   2)   Removing any leading or trailing characters from the set of whitespace characters
399   defined in the previous list item about CIM-XML producers. For compatibility with
400   implementations of earlier versions of this specification, such removal is
401   recommended for an implementation even if the addition of such leading or trailing
402   characters is not performed by that implementation.

403   **DEPRECATED**

---

404   **5.1.1.4   Escaping of element content and attribute values**

405   The provisions (that is, keywords like "shall", etc.) about escaping defined in clause 2.4 of the W3C XML
406   specification shall be interpreted to apply to CIM-XML producers.

407 CIM-XML consumers shall support the unescaping of all forms of escaping permissible for CIM-XML
408 producers, as defined in the W3C XML specification and in this specification.

409 **5.1.1.4.1   Use of XML numeric character references**

410 Clarifying clause 2.4 of the W3C XML specification, CIM-XML producers may escape any characters in
411 the content of any elements and in the value of any attributes using XML numeric character references
412 (e.g. `&` or `&`).

413 Note that this includes all UCS characters valid for the element or attribute in question, not just those that
414 may be escaped using XML entity references (see 5.1.1.4.2).

415 **5.1.1.4.2   Use of XML entity references**

416 Clarifying clause 2.4 of the W3C XML specification, CIM-XML producers may escape the respective
417 characters in the content of any elements (except when within a CDATA section) and in the value of any
418 attributes using the XML entity references defined in that clause:

419   `&`
420   `<`
421   `>`
422   `'`
423   `"`

424 **5.1.1.4.3   Use of CDATA sections**

425 Restricting clause 2.7 of the W3C XML specification, CDATA sections may be used by CIM-XML
426 producers only for representing all or part of the content of any elements defined as #PCDATA. CDATA
427 sections shall not be used in the content of any other elements or in the value of any attributes.

428 Note that some CIM-XML elements are defined as #PCDATA but do not represent string-typed CIM
429 values (for example, VALUE and KEYVALUE elements for boolean and numeric CIM values, or HOST).

430 Note that the W3C XML specification permits the use of multiple non-nested CDATA sections in the
431 content of a single element, and the use of CDATA sections that cover only a subset of the element
432 content. Using multiple CDATA sections may occur for example in the presence of nested embedded
433 instances (see 5.1.1.4.5).

434 **5.1.1.4.4   Combining different escaping mechanisms**

435 Note that the W3C XML specification permits the combined use of numeric character references, entity
436 references and CDATA sections in the content of the same element. Note that entity and character
437 references can only be combined with CDATA sections if the CDATA section covers a subset of the
438 character data and the references are used in the portion of the character data that is outside of the
439 CDATA section. Entity and character references that occur in CDATA sections are not invalid, but they
440 are interpreted as a sequence of characters, rather than a character reference.

441 Example:

442 The following VALUE element:

443   `<VALUE>XML-escaped: & < & <![CDATA[CDATA section escaped: & <`
444   `&]]></VALUE>`

445 has the following CIM value after unescaping:

446   `XML-escaped: & < & CDATA section escaped: & < &`

5.1.1.4.5   Nested embedded objects

448 An embedded object in the value of a property in an embedded instance is termed "nested embedded
449 object". Such nesting may occur with arbitrary depth, whereby the non-leaf levels always are embedded
450 instances and the leaf level may be an embedded instance or an embedded class.

451 Each level of nested embedded objects shall be escaped separately, treating the result of the previous
452 escaping as input to the next level of escaping. What is escaped at each level, is always the value of the
453 string-typed property defined as the embedded object or embedded instance, that is, the content of the
454 VALUE element representing that property value, as described in 5.3.3.1.1.

455 Nested escaping using XML numeric character references (see 5.1.1.4.1) or entity references (see
456 5.1.1.4.2) works automatically by applying the escaping rules on the string-typed property value
457 representing the embedded object.

458 Escaping using CDATA sections (see 5.1.1.4.3) requires a specific approach in order for it to work with
459 nested embedded objects, because CDATA sections cannot simply be nested within each other. The
460 character data that needs to be escaped again in context of an outer embedded instance already
461 contains the CDATA section of the inner embedded object. One approach that works in this situation is to
462 use two adjacent CDATA sections for the outer escaping that split each end marker (`]]>`) of the inner
463 CDATA section such that its first portion (for example `]]`) goes into the first (outer) CDATA section and
464 its remainder (in this case `>`) goes into the second (outer) CDATA section. This prevents the end parker
465 of an inner CDATA section to end an outer CDATA section. The unchanged start marker of an inner
466 CDATA section does not hurt, because once in the outer CDATA section, the inner start marker is
467 encountered but is treated as normal character data.

468 Example with a nesting level of two:

469   Class definitions in MOF:

```
470        class A {
471          [EmbeddedInstance("B")]
472           string InstanceOfB;
473        };
474
475        class B {
476          [EmbeddedInstance("C")]
477           string InstanceOfC;
478        };
479
480        class C {
481            string PropC;
482        };
```

483   Representation of an instance of class A using escaping with XML entity references. The color
484   scheme indicates the levels of nesting. One can see that the number of nested applications of XML-
485   escaping matches the nesting level of the instances:

```
486         <INSTANCE CLASSNAME="A">
487           ...
488          <PROPERTY NAME="InstanceOfB" TYPE="string" EmbeddedObject="instance">
489           <VALUE>
490              &lt;INSTANCE CLASSNAME="B"&gt;
491                ...
492              &lt;PROPERTY NAME="InstanceOfC" TYPE="string"
493                         EmbeddedObject="instance"&gt;
494                &lt;VALUE&gt;
495                   &amp;lt;INSTANCE CLASSNAME="B"&amp;gt;
496                     ...
497                   &amp;lt;PROPERTY NAME="PropC" TYPE="string"&amp;gt;
498                     &amp;lt;VALUE&amp;gt;a string&amp;lt;/VALUE&amp;gt;
499                   &amp;lt;/PROPERTY&amp;gt;
500                     ...
501                   &amp;lt;/INSTANCE&amp;gt;
502                &lt;/VALUE&gt;
503              &lt;/PROPERTY&gt;
504                ...
505              &lt;/INSTANCE&gt;
506           </VALUE>
507          </PROPERTY>
508           ...
509         </INSTANCE>
```

510 Representation of an instance of class A using escaping with CDATA sections. One can see that the
511 usage of the CDATA section for escaping the embedded instance of B in property A.InstanceOfB
512 uses one simple application of CDATA escaping. However, an end marker was found in the data to
513 be escaped. Therefore, two adjacent CDATA sections are used when escaping the value of property
514 A.InstanceofB, whose boundary cuts the inner end marker into two parts.

```
515      <INSTANCE CLASSNAME="A">
516        ...
517        <PROPERTY NAME="InstanceOfB" TYPE="string" EmbeddedObject="instance">
518          <VALUE>
519            <![CDATA[                             <- start marker of first outer
520                                                     CDATA section
521              <INSTANCE CLASSNAME="B">
522                ...
523                <PROPERTY NAME="InstanceOfC" TYPE="string"
524                        EmbeddedObject="instance">
525                  <VALUE>
526                    <![CDATA[                     <- start marker of inner CDATA
527                                                     section
528                      <INSTANCE CLASSNAME="B">
529                        ...
530                        <PROPERTY NAME="PropC" TYPE="string">
531                          <VALUE>a string</VALUE>
532                        </PROPERTY>
533                        ...
534                      </INSTANCE>
535                                                     end marker of inner CDATA
536                      ↓↓              ↓          <- section, cut into two parts
537                      ]]]]><![CDATA[>
538                      ↑↑↑↑↑↑↑↑↑↑↑↑↑              <- end marker of first and
539                                                     start marker of second
540                                                     outer CDATA section
541                  </VALUE>
542                </PROPERTY>
543                ...
544              </INSTANCE>
545            ]]>                                   <- end marker of second outer
546                                                     CDATA section
547          </VALUE>
548        </PROPERTY>
549        ...
550      </INSTANCE>
```

551   5.1.1.4.6 Requirements for escaping

552 For better interoperability, the following rules on escaping apply to CIM-XML producers, in addition to the
553 rules defined in the W3C XML specification:

554 • UCS characters in the range of U+0000 to U+001F should be escaped, using any of the
555   escaping mechanisms described in 5.1.1.

5.1.1.5 Character Repertoire

557 The following rules about the repertoire of UCS characters apply for representing the CIM-XML payload:

558 • If the XML declaration of the CIM-XML payload specifies version="1.0":

559 – The valid UCS characters in the range of U+0020 to U+10FFFF may be used.
560 – The UCS characters U+0009, U+000A, U+000D may be used.
561 – The other UCS characters in the range of U+0000 to U+001F shall not be used.

562 • If the XML declaration of the CIM-XML payload specifies version="1.1":

563 – The valid UCS characters in the range of U+0001 to U+10FFFF may be used.
564 – The UCS character U+0000 shall not be used.

565 Note that these rules are consistent with the W3C XML specification; XML 1.0 only supports
566 representation of the UCS characters U+0009, U+000A, U+000D in the range of U+0000 to U+001F.
567 XML 1.1 extends the support for that range to all characters except U+0000.

568 Note that DSP0004 permits string and char16 typed values to use all UCS characters in the range of
569 U+0000 to U+001F. Thus, CIM-XML supports only a subset of that range, depending on the XML version
570 used.

5.2 Entity Descriptions

572 This subclause describes each of the parameter entities used in the CIM-XML schema vocabulary. The
573 use of parameter entities has been adopted to highlight common features of the DTD.

5.2.1 CIMName

575 The CIMName entity describes the name of a CIM element (class, instance, method, property, qualifier,
576 or parameter). The value shall be a legal CIM element name (DSP0004).

577
```
<!ENTITY % CIMName "NAME  CDATA  #REQUIRED">
```

5.2.2 CIMType

579 The CIMType entity describes the allowed type descriptions for a non-reference CIM property, CIM
580 qualifier, or non-reference CIM method parameter.

581
```
<!ENTITY % CIMType "TYPE
    (boolean | string | char16 | uint8 | sint8 | uint16 | sint16 | uint32 |
    sint32 | uint64 | sint64 | datetime | real32 | real64)">
```

5.2.3 QualifierFlavor

585 The QualifierFlavor entity describes the flavor settings for a CIM qualifier, modeled as XML attributes.

586 DEPRECATION NOTE: The TOINSTANCE attribute is deprecated and may be removed from the QualifierFlavor
587 entity in a future version of this document. Use of this qualifier is discouraged.

588
```
<!ENTITY % QualifierFlavor "OVERRIDABLE (true|false)  'true'
    TOSUBCLASS   (true|false)  'true'
    TOINSTANCE   (true|false)  'false'
    TRANSLATABLE (true|false)  'false'">
```

5.2.4 ClassOrigin

593 The ClassOrigin entity describes the originating class of a CIM property or method.

594 The originating class of a CIM property or method for the purpose of the ClassOrigin entity is the leaf-
595 most class that defines or overrides the property or method.

596 The CLASSORIGIN attribute defines the name of the originating class of the CIM element represented by
597 the XML element to which the attribute is attached.

598   `<!ENTITY % ClassOrigin "CLASSORIGIN CDATA #IMPLIED">`

5.2.5 Propagated

599

600 The Propagated entity is a convenient shorthand for the PROPAGATED attribute, which may apply to a
601 CIM property, method, or qualifier.

602 This attribute indicates whether the definition of the CIM property, qualifier, or method is local to the CIM
603 class (respectively, instance) in which it appears, or was propagated without modification from the
604 underlying subclass (respectively, class), as defined by the DSP0004.

605   `<!ENTITY % Propagated "PROPAGATED (true|false) 'false'">`

606 Uses of the PROPAGATED attribute include:

607   • To facilitate the rendering of CIM-XML declarations into MOF syntax, which by convention only
608   describes local overrides in a CIM subclass or instance
609   • To filter XML representations of CIM classes or instances so that they can be returned as
610   responses to CIM operation requests (DSP0200), which require only local elements

5.2.6 ArraySize

611

612 The ArraySize entity is a convenient shorthand for the ARRAYSIZE attribute.

613   `<!ENTITY % ArraySize "ARRAYSIZE CDATA #IMPLIED">`

614 The ARRAYSIZE attribute defines the size of the array when it is a fixed-length array (see DSP0004).
615 The value of this attribute (if it is present) shall be a positive integer.

5.2.7 SuperClass

616

617 The SuperClass entity is a convenient shorthand for the SUPERCLASS attribute.

618   `<!ENTITY % SuperClass "SUPERCLASS CDATA #IMPLIED">`

619 This attribute defines the name of the superclass. Where it is omitted, it shall be inferred that the owning
620 element has no superclass.

5.2.8 ClassName

621

622 The ClassName entity is a convenient shorthand for the CLASSNAME attribute. The value shall be a
623 legal CIM class name (DSP0004).

624   `<!ENTITY % ClassName "CLASSNAME CDATA #REQUIRED">`

5.2.9 ReferenceClass

625

626 The ReferenceClass entity is a convenient shorthand for the REFERENCECLASS attribute. If this entity is
627 present, the value shall be a legal CIM class name (DSP0004).

628   `<!ENTITY % ReferenceClass "REFERENCECLASS CDATA #IMPLIED">`

629 The value defines the class name for the reference, and the requirement for the existence of this attribute
630 depends on the element in which it is used. The expected behavior is that the REFERENCECLASS
631 attribute shall exist for classes and should not exist for instances.

632 5.2.10 ParamType

633 The ParamType entity describes the allowed type descriptions for parameter values or return values.

634   `<!ENTITY % ParamType "PARAMTYPE`
635   `(boolean | string | char16 | uint8 | sint8 | uint16 | sint16 | uint32 |`
636   `sint32 | uint64 | sint64 | datetime | real32 | real64 | reference |`
637   `object | instance)">`

638 **DEPRECATED:** The values `"object"` and `"instance"` have been deprecated in version 2.4.0 of this
639 specification because they are not used.

640 5.2.11 EmbeddedObject

641 The EmbeddedObject entity defines an embedded object or an embedded instance. This entity may be
642 applied only to entities that have the Type string.

643   `<!ENTITY % EmbeddedObject "(object | instance) #IMPLIED">`

644 This attribute is to be used to represent the existence of an EMBEDDEDINSTANCE or
645 EMBEDDEDOBJECT qualifier on the corresponding metadata (method, parameter, or property).

646 If the EMBEDDEDOBJECT qualifier is defined for the method, parameter, or property, the
647 EmbeddedObject attribute shall be attached to the corresponding PROPERTY in any instance,
648 PARAMVALUE, or RETURNVALUE with the value "object".

649 If the EMBEDDEDINSTANCE qualifier exists for the method, parameter, or property, the
650 EmbeddedObject attribute shall be attached to the corresponding PROPERTY in any instance,
651 PARAMVALUE, or RETURNVALUE with the value "instance".

652 5.3   Element Descriptions

653 This subclause describes each of the elements in the CIM-XML schema.

654 5.3.1   Top-Level Element: CIM

655 The CIM element is the root element of every XML document that is valid with respect to this schema.

656 Each document takes one of two forms: it contains a single MESSAGE element that defines a CIM-XML
657 message (to be used in DSP0200), or it contains a DECLARATION element that is used to declare a set
658 of CIM objects.

659   `<!ELEMENT CIM (MESSAGE | DECLARATION)>`
660   `<!ATTLIST CIM`
661   `CIMVERSION CDATA #REQUIRED`
662   `DTDVERSION CDATA #REQUIRED>`

663 The CIMVERSION attribute defines the version of the DSP0004 to which the XML document conforms. It
664 shall be in the form of "M.N.U", where M is the major version of the specification, N is the minor version of
665 the specification, and U is the update version of the specification, each in their decimal representation
666 without leading zeros. Any draft letter in the version of the specification shall not be represented in the

667   attribute (for example, 2.3.0, 2.4.0). Implementations need to validate only the major version, as all minor
668   and update versions are backward compatible. Implementations may look at the minor or update version
669   to determine additional capabilities.

670   The DTDVERSION attribute defines the version of DSP0201 (this document) to which the XML document
671   conforms. It shall be in the form of "M.N.U", where M is the major version of the specification, N is the
672   minor version of the specification, and U is the update version of the specification, each in their decimal
673   representation without leading zeros. Any draft letter in the version of the specification shall not
674   be represented in the attribute (for example, 2.2.0, 2.3.0). Implementations need to validate only the
675   major version, as all minor and update versions are backward compatible. Implementations may look at
676   the minor or update version to determine additional capabilities.

5.3.2 Declaration Elements

678   This subclause defines those XML elements that are concerned with expressing the declaration of CIM
679   objects.

5.3.2.1 DECLARATION

681   The DECLARATION element defines a set of one or more declarations of CIM objects. These are
682   partitioned into logical declaration subsets.

683   `<!ELEMENT DECLARATION`
684   `(DECLGROUP | DECLGROUP.WITHNAME | DECLGROUP.WITHPATH)+>`

5.3.2.2 DECLGROUP

686   The DECLGROUP element defines a set of CIM class, instance, and qualifier declarations. It may
687   optionally include a NAMESPACEPATH or LOCALNAMESPACEPATH element, which, if present, defines
688   the common namespace in which all objects within the group are declared.

689   The objects within the group are CIM classes, instances, and qualifiers.

690   `<!ELEMENT DECLGROUP`
691   `((LOCALNAMESPACEPATH|NAMESPACEPATH)?, QUALIFIER.DECLARATION*, VALUE.OBJECT*)>`

5.3.2.3 DECLGROUP.WITHNAME

693   The DECLGROUP.WITHNAME element defines a set of CIM class, instance, and qualifier declarations. It
694   may optionally include a NAMESPACEPATH or LOCALNAMESPACEPATH element, which, if present,
695   defines the common namespace in which all objects within the group are declared.

696   The objects within the group are CIM classes, instances, and qualifiers.

697   The DECLGROUP.WITHNAME element extends the DECLGROUP element in the sense that any
698   instance declaration contains an explicit instance name (that is, a model path in the terms of DSP0004).

699   `<!ELEMENT DECLGROUP.WITHNAME`
700   `((LOCALNAMESPACEPATH | NAMESPACEPATH)?, QUALIFIER.DECLARATION*,`
701   `VALUE.NAMEDOBJECT*)>`

5.3.2.4 DECLGROUP.WITHPATH

703   The DECLGROUP.WITHPATH element defines a set of CIM class and instance declarations. Each object
704   is declared with its own independent naming and location information.

705   `<!ELEMENT DECLGROUP.WITHPATH`
706   `(VALUE.OBJECTWITHPATH | VALUE.OBJECTWITHLOCALPATH)*>`

707 **5.3.2.5 QUALIFIER.DECLARATION**

708 The QUALIFIER.DECLARATION element defines a single CIM qualifier declaration.

709 A VALUE or a VALUE.ARRAY subelement shall be present if the qualifier declaration has a non-NULL
710 default value defined. A VALUE subelement is used if the qualifier has a non-array type. A
711 VALUE.ARRAY subelement is used if the qualifier has an array type. Absence of the VALUE and
712 VALUE.ARRAY subelements shall be interpreted as a default value of NULL.

713 The SCOPE subelement, if present, defines the valid set of scopes for this qualifier. Absence of the
714 SCOPE subelement implies that there is no restriction on the scope at which the qualifier may be applied
715 (so that it has "any" scope in the terminology of DSP0004).

```
716     <!ELEMENT QUALIFIER.DECLARATION
717         (SCOPE?, (VALUE | VALUE.ARRAY)?)>
718     <!ATTLIST QUALIFIER.DECLARATION
719         %CIMName;
720         %CIMType;                #REQUIRED
721         ISARRAY    (true|false)  #IMPLIED
722         %ArraySize;
723         %QualifierFlavor;>
```

724 The NAME attribute (defined by the CIMName entity) defines the name of the qualifier, and the TYPE
725 attribute (defined by the CIMType entity) and ISARRAY attributes together define the CIM type.

726 The ISARRAY attribute shall be present if the qualifier declares no default value, in order to infer whether
727 the qualifier has an array type. The ISARRAY attribute should be absent if the qualifier declares a non-
728 NULL default value; in this case, whether the qualifier has an array type can be deduced from whether a
729 VALUE or VALUE.ARRAY element is used to declare that default. If the ISARRAY attribute is present, its
730 value shall be consistent with the declared qualifier default value.

731 The ARRAYSIZE attribute (defined by the ArraySize entity) shall not be present. Its use on this element
732 has been deprecated in version 2.4 of this document. Note that DSP0004 defines that qualifier types that
733 are arrays need to be variable-length arrays.

734 The flavor attributes declared using the QualifierFlavor entity define the propagation and override
735 semantics for the qualifier.

736 **5.3.2.6 SCOPE**

737 The SCOPE element defines the scope of a QUALIFIER.DECLARATION when there are restrictions on
738 the scope of the qualifier declaration.

```
739     <!ELEMENT SCOPE EMPTY>
740     <!ATTLIST SCOPE
741         CLASS       (true|false)   "false"
742         ASSOCIATION (true|false)   "false"
743         REFERENCE   (true|false)   "false"
744         PROPERTY    (true|false)   "false"
745         METHOD      (true|false)   "false"
746         PARAMETER   (true|false)   "false"
747         INDICATION  (true|false)   "false">
```

748 The attributes define which scopes are valid. A SCOPE element shall declare at least one attribute with a
749 `true` value. (Otherwise, the qualifier would have no applicable scope.)

750 **5.3.3   Value Elements**

751 This subclause defines those XML elements that are concerned with expressing CIM-typed values.

752 **5.3.3.1   VALUE**

753 The VALUE element is used to define a single (non-array), non-reference, non-NULL CIM value.

754   `<!ELEMENT VALUE (#PCDATA)>`

755 Because the same element is used for values of all CIM types, the CIM type determines the format of the
756 content of the VALUE element, as defined in the following subclauses of this clause. These subclauses
757 also define when the VALUE element is character-preserving (see 5.1.1.2). In most cases, the CIM type
758 is provided using the `TYPE` or `PARAMTYPE` attributes of the direct or an indirect parent element of the
759 VALUE element.

760 **5.3.3.1.1   String Values**

761 If the CIM type is `string`, the content of the VALUE element shall be a sequence of zero or more UCS
762 characters that represent the CIM value. An empty content of the VALUE element value represents an
763 empty string (that is, "" in MOF). The character repertoire defined for the CIM string type shall be
764 supported as defined in [DSP0004](#) and 5.1.1.5. The content of the VALUE element shall not have
765 additional surrounding string delimiter characters (such as double-quote or single-quote characters)
766 compared to the CIM value. The actual representation of UCS characters depends on the `encoding`
767 attribute defined in the XML declaration (`<?xml ... ?>`).

768 This use of the VALUE element is character-preserving (see 5.1.1.2).

769 **5.3.3.1.2   Character Values**

770 If the CIM type is `char16`, the content of the VALUE element shall be a single UCS character that
771 represents the CIM value. The character repertoire defined for the CIM char16 type shall be supported as
772 defined in [DSP0004](#) and 5.1.1.5. The content of the VALUE element shall not have additional surrounding
773 string delimiter characters (such as double-quote or single-quote characters) compared to the CIM value.
774 The actual representation of the UCS character depends on the `encoding` attribute defined in the XML
775 declaration (`<?xml ... ?>`).

776 This use of the VALUE element is character-preserving (see 5.1.1.2).

777 **5.3.3.1.3   Real Values**

778 If the CIM type is `real32` or `real64`, the content of the VALUE element shall conform to the format
779 defined by the following ABNF rules and shall represent the CIM value, where `decimalDigit` is any
780 character from the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}:

781   `[ "+" / "-" ] *decimalDigit "." 1*decimalDigit [ ( "e" / "E" ) [ "+" / "-" ]`
782   `1*decimalDigit ] / specialState`
783
784   `specialState = "INF" / "-INF" / "NaN"`

785 The basis for the exponent shall be 10. The significand shall be represented with a precision of at least 9
786 decimal digits for real32 and at least 17 digits for real64. Trailing zeros in the fractional part and leading
787 zeros in the whole part of the significand may be omitted. Leading zeros in the exponent may be omitted.

788 NOTE: This definition of a minimum precision guarantees that the value of CIM real types in their binary
789 representation (defined by IEEE 754) does not change when converting it to the decimal representation and back to
790 the binary representation.

791 The special states for floating point numbers defined by IEEE 754, +Infinity, -Infinity, and the NaN states,
792 shall be represented by the literals "`INF`", "`-INF`", and "`NaN`", respectively (consistent with the XML
793 datatypes `xs:float` and `xs:double` defined in XML Schema, Part 2). The NaN states shall all be
794 represented by the same string, "`NaN`". These literals shall be produced with the lexical case as stated;
795 they shall be consumed using case insensitive parsing, for backward compatibility with existing
796 implementations.

797 This use of the VALUE element is whitespace-tolerant (see 5.1.1.3).

5.3.3.1.4   Boolean Values

799 If the CIM type is `boolean`, the content of the VALUE element shall be either `TRUE` or `FALSE` and shall
800 represent the CIM value. These values shall be treated as case-insensitive by CIM-XML consumers. CIM-
801 XML producers should use upper case.

802 This use of the VALUE element is whitespace-tolerant (see 5.1.1.3).

5.3.3.1.5   Integer Values

804 If the CIM type belongs to the set {`uint8`, `uint16`, `uint32`, `uint64`}, the content of the VALUE element
805 shall be a valid unsigned decimal or hexadecimal value that represents the CIM value.

806 If the CIM type belongs to the set {`sint8`, `sint16`, `sint32`, `sint64`}, the content of the VALUE element
807 shall be a valid signed decimal or hexadecimal value that represents the CIM value.

808 Decimal values have the format defined by the following ABNF rule, where `decimalDigit` is any
809 character from the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} and `positiveDecimalDigit` is any decimal digit
810 other than 0:

811   `[ "+" / "-" ] ( positiveDecimalDigit *decimalDigit / "0" )`

812 The leading sign character shall not be used when the CIM type is unsigned.

813 Hexadecimal values have the format defined by the following ABNF rule, where `hexDigit` is either a
814 `decimalDigit` or a character from the set {a, A, b, B, c, C, d, D, e, E, f, F}:

815   `[ "+" / "-" ] ( "0x" / "0X" ) 1*hexDigit`

816 The leading sign character shall not be used when the CIM type is unsigned.

817 This use of the VALUE element is whitespace-tolerant (see 5.1.1.3).

5.3.3.1.6   Datetime Values

819 If the CIM type is `datetime`, the content of the VALUE element shall be a valid datetime value that
820 represents the CIM value, as defined in detail by DSP0004. (For interval values, the format is
821 `ddddddddhhmmss.mmmmmm:000`; for absolute values, the format is `yyyymmddhhmmss.mmmmmmsutc`.)

822 The value shall not be surrounded by string delimiter characters (such as double-quote or single-quote
823 characters).

824 This use of the VALUE element is character-preserving (see 5.1.1.2).

825 NOTE: This use of the VALUE element needs to be character-preserving in order to properly handle the
826 case where a key property has datetime type and its TYPE attribute is not provided (see KEYVALUE).

5.3.3.2   VALUE.ARRAY

828 The VALUE.ARRAY element is used to represent a CIM value of array type.

---

829 CIM arrays are classified as "Bag", "Ordered", or "Indexed" (refer to DSP0004) using the ARRAYTYPE
830 qualifier. If the array is Ordered or Indexed, the subelements of VALUE.ARRAY shall appear in the order
831 of the array entries.

832 If the value of an array entry is NULL, the VALUE.NULL subelement shall be used to represent the array
833 entry. Otherwise, the VALUE subelement shall be used.

834 NOTE: For string datatypes, a VALUE element with an empty PCDATA value indicates an empty string (that is, "").

835   `<!ELEMENT VALUE.ARRAY (VALUE | VALUE.NULL)*>`

5.3.3.3   VALUE.REFERENCE

837 The VALUE.REFERENCE element is used to define a single CIM reference property value.

838 If a LOCALCLASSPATH or LOCALINSTANCEPATH subelement is used, the target object is assumed to
839 be on the same host. If a CLASSNAME or INSTANCENAME subelement is used, the target object is
840 assumed to be in the same namespace.

841   `<!ELEMENT VALUE.REFERENCE`
842   `    (CLASSPATH | LOCALCLASSPATH | CLASSNAME | INSTANCEPATH | LOCALINSTANCEPATH |`
843   `    INSTANCENAME)>`

5.3.3.4   VALUE.REFARRAY

845 The VALUE.REFARRAY element is used to represent the value of an array of CIM references.

846 CIM arrays are classified as "Bag", "Ordered", or "Indexed" (refer to DSP0004) using the ARRAYTYPE
847 qualifier. If the array is Ordered or Indexed, the subelements shall appear in the order of the array entries.

848 If the value of an array entry is NULL, the VALUE.NULL subelement shall be used to represent the array
849 entry. Otherwise, the VALUE.REFERENCE subelement shall be used.

850   `<!ELEMENT VALUE.REFARRAY (VALUE.REFERENCE | VALUE.NULL)*>`

5.3.3.5   VALUE.OBJECT

852 The VALUE.OBJECT element is used to define a value that comprises a single CIM class or instance
853 definition.

854   `<!ELEMENT VALUE.OBJECT (CLASS | INSTANCE)>`

5.3.3.6   VALUE.NAMEDINSTANCE

856 The VALUE.NAMEDINSTANCE element is used to define a value that comprises a single named CIM
857 instance definition.

858   `<!ELEMENT VALUE.NAMEDINSTANCE (INSTANCENAME, INSTANCE)>`

5.3.3.7   VALUE.NAMEDOBJECT

860 The VALUE.NAMEDOBJECT element is used to define a value that comprises a single named CIM class
861 or instance definition.

862   `<!ELEMENT VALUE.NAMEDOBJECT (CLASS |(INSTANCENAME, INSTANCE))>`

5.3.3.8   VALUE.OBJECTWITHPATH

864 The VALUE.OBJECTWITHPATH element is used to define a value that comprises a single CIM object
865 (class or instance) definition with additional information that defines the absolute path to that object.

866   `<!ELEMENT VALUE.OBJECTWITHPATH ((CLASSPATH, CLASS) | (INSTANCEPATH, INSTANCE))>`

5.3.3.9   VALUE.OBJECTWITHLOCALPATH

867

868 The VALUE.OBJECTWITHLOCALPATH element is used to define a value that comprises a single CIM
869 object (class or instance) definition with additional information that defines the local path to that object.

870   `<!ELEMENT VALUE.OBJECTWITHLOCALPATH`
871   `((LOCALCLASSPATH, CLASS) | (LOCALINSTANCEPATH, INSTANCE))>`

5.3.3.10 VALUE.NULL

872

873 The VALUE.NULL element is used to represent a NULL value.

874 NOTE: In some cases, omission of a subelement indicates the NULL value, instead of using VALUE.NULL.

875   `<!ELEMENT VALUE.NULL EMPTY>`

5.3.3.11 VALUE.INSTANCEWITHPATH

876

877 The VALUE.INSTANCEWITHPATH element is used to define a value that comprises a single CIM
878 instance definition with additional information that defines the absolute path to that object.

879   `<!ELEMENT VALUE.INSTANCEWITHPATH (INSTANCEPATH, INSTANCE)>`

5.3.4   Naming and Location Elements

880

881 This clause defines those XML elements that are concerned with expressing the name and location of
882 CIM objects (namespaces, classes, instances and qualifiers).

5.3.4.1   NAMESPACEPATH

883

884 The NAMESPACEPATH element is used to define a namespace path. It consists of a HOST element and
885 a LOCALNAMESPACEPATH element.

886 The NAMESPACE elements shall appear in hierarchy order, with the root namespace appearing first.

887   `<!ELEMENT NAMESPACEPATH (HOST, LOCALNAMESPACEPATH)>`

5.3.4.2   LOCALNAMESPACEPATH

888

889 The LOCALNAMESPACEPATH element is used to define a local namespace path (one without a host
890 component). It consists of one or more NAMESPACE elements (one for each namespace in the path).

891   `<!ELEMENT LOCALNAMESPACEPATH (NAMESPACE+)>`

5.3.4.3   HOST

892

893 The HOST element is used to define a single host, optionally including a port number.

894   `<!ELEMENT HOST (#PCDATA)>`

895 The format of the content of the HOST element shall conform to the following ABNF rule:

896   `hostport = host [ ":" port ]`

897 Where `host` and `port` are ABNF rules defined in RFC3986.

898 If port is not specified, the CIM-XML consumer shall assume the port numbers registered with IANA for
899 the CIM-XML protocol as defaults:

900   • port 5988, for use with CIM-XML over HTTP
901   • port 5989, for use with CIM-XML over HTTPS

902   This specification defines the following additional rules for using the ABNF rules from RFC3986 for the
903   content of the HOST element:

904   • `host` (and `port`) shall not use URI percent-encoding
905   • `reg-name` (used to specify hostnames) is restricted to conform to the syntax for DNS domain
906   names as defined in section 3.1 of RFC1034 (that is, segments are separated by a dot, each
907   segment is limited to 63 characters, and the total length is limited to 255 characters)

908   Note that specifying zone identifiers (also known as zone indices) for IPv6 addresses does not make
909   sense in IPv6 addresses that are transmitted in a protocol to another host, because their meaning is
910   strictly local to the originating host. For this reason, the syntax defined by the `IPv6address` ABNF rule
911   from RFC3986 (which does not permit the use of zone identifiers) is sufficient.

912   Implementations shall support the specification of port, and the syntax defined by `IPv4address` and
913   `reg-name`. Implementations should in addition support the syntax defined by `IPv6address`.
914   Implementations do not need to support the syntax defined by the `IPFuture` ABNF rule from RFC3986
915   at this point.

916   **5.3.4.4   NAMESPACE**

917   The NAMESPACE element is used to define a single namespace component of a namespace path.

918   `<!ELEMENT NAMESPACE EMPTY>`
919   `<!ATTLIST NAMESPACE`
920   `%CIMName;>`

921   The NAME attribute (defined by the CIMName entity) defines the name of the namespace.

922   **5.3.4.5   CLASSPATH**

923   The CLASSPATH element defines the absolute path to a CIM class. It is formed from a namespace path
924   and class name.

925   `<!ELEMENT CLASSPATH (NAMESPACEPATH, CLASSNAME)>`

926   **5.3.4.6   LOCALCLASSPATH**

927   The LOCALCLASSPATH element defines the local path to a CIM class. It is formed from a local
928   namespace path and class name.

929   `<!ELEMENT LOCALCLASSPATH (LOCALNAMESPACEPATH, CLASSNAME)>`

930   **5.3.4.7   CLASSNAME**

931   The CLASSNAME element defines the qualifying name of a CIM class.

932   `<!ELEMENT CLASSNAME EMPTY>`
933   `<!ATTLIST CLASSNAME`
934   `%CIMName;>`

935   The NAME attribute (defined by the CIMName entity) defines the name of the class.

936 **5.3.4.8 INSTANCEPATH**

937 The INSTANCEPATH element defines the absolute path to a CIM instance. It comprises a namespace
938 path and an instance name (model path).

939   `<!ELEMENT INSTANCEPATH (NAMESPACEPATH, INSTANCENAME)>`

940 **5.3.4.9 LOCALINSTANCEPATH**

941 The LOCALINSTANCEPATH element defines the local path to a CIM instance. It comprises a local
942 namespace path and an instance name (model path).

943   `<!ELEMENT LOCALINSTANCEPATH (LOCALNAMESPACEPATH, INSTANCENAME)>`

944 **5.3.4.10 INSTANCENAME**

945 The INSTANCENAME element defines the location of a CIM instance within a namespace (it is referred
946 to in DSP0004 as a model path). It comprises a class name and key-binding information.

947 If the class has a single key property, a single KEYVALUE or VALUE.REFERENCE subelement may be
948 used to describe the (necessarily) unique key value without a key name. Alternatively, a single
949 KEYBINDING subelement may be used instead.

950 If the class has more than one key property, a KEYBINDING subelement shall appear for each key.

951 If no key-bindings are specified, the instance is assumed to be a singleton instance of a keyless class.

952   `<!ELEMENT INSTANCENAME (KEYBINDING* | KEYVALUE? | VALUE.REFERENCE?)>`
953   `<!ATTLIST INSTANCENAME`
954   `%ClassName;>`

955 The CLASSNAME attribute (defined by the ClassName entity) defines the name of the class for this path.

956 **5.3.4.11 OBJECTPATH**

957 The OBJECTPATH element is used to define a full path to a single CIM object (class or instance).

958   `<!ELEMENT OBJECTPATH (INSTANCEPATH | CLASSPATH)>`

959 **5.3.4.12 KEYBINDING**

960 The KEYBINDING element defines a single key property value binding.

961   `<!ELEMENT KEYBINDING (KEYVALUE | VALUE.REFERENCE)>`
962   `<!ATTLIST KEYBINDING`
963   `%CIMName;>`

964 The NAME attribute (defined by the CIMName entity) indicates the name of the key property.

965 **5.3.4.13 KEYVALUE**

966 The KEYVALUE element defines the value of a (scalar) key property that has a non-reference type.

```
967      <!ELEMENT KEYVALUE (#PCDATA)>
968      <!ATTLIST KEYVALUE
969          VALUETYPE    (string | boolean | numeric) "string"    (DEPRECATED)
970          %CIMType;    #REQUIRED>
```

971 The VALUETYPE attribute provides information regarding the data type to allow the transformation of the
972 key value to and from its textual equivalent (as part of a text-based CIM object path, for example as
973 defined in DSP0207). The value of this attribute shall conform to the following rules:

974   • If the CIM type is string, datetime, or char16, the value is string.
975   • If the CIM type is boolean, the value is boolean.
976   • Otherwise, the value is numeric.

977 The VALUETYPE attribute has been deprecated in version 1.4 of this document. Use the TYPE attribute
978 instead.

979 The TYPE attribute (defined by the CIMType entity) identifies the CIM type of the key property. The TYPE
980 attribute is required to be provided by CIM-XML producers as of version 1.4 of this document. For
981 implementations of earlier versions of this document, it is strongly recommended that CIM-XML producers
982 always provide the TYPE attribute, because it supports strongly typed representations of values in CIM-
983 XML consumers and can be used to improve performance.

984 The content of the KEYVALUE element represents the property value. Note that key properties cannot be
985 NULL. Because the KEYVALUE element is used for key property values of all CIM types, the CIM type of
986 the key property determines the format of the content of the KEYVALUE element, as follows:

987   • CIM-XML producers shall set the content of the KEYVALUE element based on the CIM type of
988   the key property as defined in 5.3.3.1.
989   • If the TYPE attribute is provided, CIM-XML consumers shall interpret the content of the
990   KEYVALUE element based on the TYPE attribute as defined in 5.3.3.1.
991   • If the TYPE attribute is not provided (for example, when earlier versions of this specification are
992   implemented), CIM-XML consumers that have no knowledge about the CIM type of the key
993   property shall interpret the content of the KEYVALUE element based on the VALUETYPE
994   attribute as follows:

995   – If the value of the VALUETYPE attribute is string, the content of the KEYVALUE
996   element shall be interpreted as defined in 5.3.3.1.1.
997   – If the value of the VALUETYPE attribute is boolean, the content of the KEYVALUE
998   element shall be interpreted as defined in 5.3.3.1.4.
999   – If the value of the VALUETYPE attribute is numeric, the content of the KEYVALUE
1000   element shall be interpreted as defined in 5.3.3.1.3 or 5.3.3.1.5, depending on which
1001   syntax matches.

5.3.5   Object Definition Elements

1003 This subclause defines those XML elements that are concerned with expressing the declaration of CIM
1004 objects (classes, instances, and qualifiers) and their components (properties, methods, and parameters).

5.3.5.1   CLASS

1006 The CLASS element defines a single CIM class.

```
1007     <!ELEMENT CLASS
1008         (QUALIFIER*, (PROPERTY | PROPERTY.ARRAY | PROPERTY.REFERENCE)*, METHOD*)>
1009     <!ATTLIST CLASS
1010         %CIMName;
1011         %SuperClass;>
```

1012   The NAME attribute (defined by the CIMName entity) defines the name of the class.

1013   The SUPERCLASS attribute (defined by the SuperClass entity), if present, defines the name of the
1014   superclass of this class. If this attribute is absent, it should be inferred that the class in question has no
1015   superclass.

5.3.5.2   INSTANCE

1017   The INSTANCE element defines a single CIM instance of a CIM class.

1018   The instance shall contain only properties defined in or inherited by the CIM class. Not all these
1019   properties are required to be present in an instance. (This is in accordance with the requirement that CIM
1020   instances have all properties defined in or inherited by the CIM class, because an <INSTANCE> is only a
1021   copied representation of the CIM instance, in a particular context). Specifications using the mapping
1022   defined in this document shall define the rules for any properties that are not present.

```
1023       <!ELEMENT INSTANCE
1024           (QUALIFIER*, (PROPERTY | PROPERTY.ARRAY | PROPERTY.REFERENCE)*)>
1025       <!ATTLIST INSTANCE
1026           %ClassName;
1027           xml:lang    NMTOKEN  #IMPLIED>
```

1028   The CLASSNAME attribute (defined by the ClassName entity) defines the name of the CIM class of which
1029   this is an instance.

5.3.5.3   QUALIFIER

1031   The QUALIFIER element defines a single CIM qualifier. If the qualifier has a non-array type, it contains a
1032   single VALUE element that represents the value of the qualifier. If the qualifier has an array type, it
1033   contains a single VALUE.ARRAY element to represent the value.

1034   If the qualifier has no assigned value (that is, it was specified without a value), the VALUE and
1035   VALUE.ARRAY subelements shall be absent. DSP0004 defines how to interpret this case, dependent on
1036   the CIM datatype.

```
1037       <!ELEMENT QUALIFIER ((VALUE | VALUE.ARRAY)?)>
1038       <!ATTLIST QUALIFIER
1039           %CIMName;
1040           %CIMType;            #REQUIRED
1041           %Propagated;
1042           %QualifierFlavor;
1043           xml:lang NMTOKEN     #IMPLIED>
```

1044   The NAME attribute (defined by the CIMName entity) defines the name of the qualifier, and the TYPE
1045   attribute (defined by the CIMType entity) defines the CIM type.

5.3.5.4   PROPERTY

1047   The PROPERTY element defines the value in a CIM instance or the definition in a CIM class of a single
1048   (non-array) CIM property that is not a reference.

1049   CIM reference properties are described using the PROPERTY.REFERENCE element.

```
1050    <!ELEMENT PROPERTY (QUALIFIER*, VALUE?)>
1051    <!ATTLIST PROPERTY
1052        %CIMName;
1053        %CIMType;              #REQUIRED
1054        %ClassOrigin;
1055        %Propagated;
1056        %EmbeddedObject;
1057        xml:lang NMTOKEN       #IMPLIED>
```

1058 A VALUE subelement shall be present if the property value or the default value of the
1059 property definition is non-NULL. Absence of the VALUE subelement shall be interpreted as a value of
1060 NULL.

1061 The NAME attribute (defined by the CIMName entity) defines the name of the property, and the TYPE
1062 attribute (defined by the CIMType entity) defines the CIM type.

1063 If the class definition for the property has the EMBEDDEDOBJECT or EMBEDDEDINSTANCE qualifier
1064 attached, the EmbeddedObject attribute (defined by the EmbeddedObject entity) shall be provided on
1065 PROPERTY elements representing properties in instances of that class, as follows. The
1066 EmbeddedObject attribute shall not be provided on PROPERTY elements representing properties in
1067 class definitions.

1068 - A property that is defined in the class as an embedded object by attaching the
1069   EMBEDDEDOBJECT qualifier on the property shall be represented using the EmbeddedObject
1070   attribute with a value of `"object"`. The (string-typed) property value shall be a valid
1071   INSTANCE element, defining a single CIM instance of a CIM class or a valid CLASS element;
1072   where these elements shall be escaped as defined in 5.1.1.4.5.
1073 - A property that is defined in the class as an embedded instance by attaching the
1074   EMBEDDEDINSTANCE qualifier on the property shall be represented using the
1075   EmbeddedObject attribute with a value of `"instance"`. The (string-typed) property value shall
1076   be a valid INSTANCE element, defining a single CIM instance of the CIM class specified in the
1077   EMBEDDEDINSTANCE qualifier; where this element shall be escaped as defined in 5.1.1.4.5.

1078 As a result, if an embedded instance has properties that are again embedded objects, each such level of
1079 embedding will be escaped separately, and thus, recursively.

5.3.5.5   PROPERTY.ARRAY

1081 The PROPERTY.ARRAY element defines the value in a CIM instance or the definition in a CIM class of a
1082 single CIM property with an array type.

1083 There is no element to model a property that contains an array of references because this is not a valid
1084 property type according to DSP0004.

```
1085    <!ELEMENT PROPERTY.ARRAY (QUALIFIER*, VALUE.ARRAY?)>
1086    <!ATTLIST PROPERTY.ARRAY
1087        %CIMName;
1088        %CIMType;              #REQUIRED
1089        %ArraySize;
1090        %ClassOrigin;
1091        %Propagated;
1092        %EmbeddedObject;
1093        xml:lang NMTOKEN       #IMPLIED>
```

1094 A [VALUE.ARRAY](#) subelement shall be present if the property value (that is, the array itself) or the default
1095 value of the property definition (that is, the array itself) is non-NULL. Absence of the [VALUE.ARRAY](#)
1096 subelement shall be interpreted as a value of NULL.

1097 The NAME attribute (defined by the [CIMName](#) entity) defines the name of the property, and the TYPE
1098 attribute (defined by the [CIMType](#) entity) defines the CIM type.

1099 On a PROPERTY.ARRAY element within a containing CLASS element, the ARRAYSIZE attribute
1100 (defined by the ArraySize entity) shall be present if the array is a fixed-length array, and shall be absent if
1101 the array is a variable-length array.

1102 On a PROPERTY.ARRAY element within a containing INSTANCE element, the ARRAYSIZE attribute
1103 should be absent, and the presence or absence of the ARRAYSIZE attribute shall not be interpreted as
1104 meaning that the property type is a fixed-length or variable-length array (that is, the [CLASS](#) definition is
1105 always authoritative in this respect).

1106 If the class definition for the property has the EMBEDDEDOBJECT or EMBEDDEDINSTANCE qualifier
1107 attached, the EmbeddedObject attribute (defined by the [EmbeddedObject](#) entity) shall be provided on
1108 PROPERTY.ARRAY elements representing properties in instances of that class, as follows. The
1109 EmbeddedObject attribute shall not be provided on PROPERTY.ARRAY elements representing
1110 properties in class definitions.

1111 • A property that is defined in the class as an embedded object by attaching the
1112 EMBEDDEDOBJECT qualifier on the property shall be represented using the EmbeddedObject
1113 attribute with a value of `"object"`. The (string-typed) property value shall be a valid
1114 [INSTANCE](#) element, defining a single CIM instance of a CIM class or a valid [CLASS](#) element;
1115 where these elements shall be escaped as defined in 5.1.1.4.5.
1116 • A property that is defined in the class as an embedded instance by attaching the
1117 EMBEDDEDINSTANCE qualifier on the property shall be represented using the
1118 EmbeddedObject attribute with a value of `"instance"`. The (string-typed) property value shall
1119 be a valid [INSTANCE](#) element, defining a single CIM instance of the CIM class specified in the
1120 EMBEDDEDINSTANCE qualifier; where this element shall be escaped as defined in 5.1.1.4.5.

1121 As a result, if an embedded instance has properties that are again embedded objects, each such level of
1122 embedding will be escaped separately, and thus, recursively.

5.3.5.6 PROPERTY.REFERENCE

1124 The PROPERTY.REFERENCE element defines the value in a CIM instance or the definition in a CIM
1125 class of a single CIM property with reference semantics. In the future, the features of [XML Linking](#) may be
1126 used to identify linking elements within the XML document.

```
1127     <!ELEMENT PROPERTY.REFERENCE (QUALIFIER*, VALUE.REFERENCE?)>
1128     <!ATTLIST PROPERTY.REFERENCE
1129         %CIMName;
1130         %ReferenceClass;
1131         %ClassOrigin;
1132         %Propagated;>
```

1133 The [VALUE.REFERENCE](#) subelement shall be present if the property value or the default value of the
1134 property definition is non-NULL. Absence of the VALUE.REFERENCE subelement shall be interpreted as
1135 a value of NULL.

1136 The NAME attribute (defined by the [CIMName](#) entity) defines the name of the property.

1137 The REFERENCECLASS attribute (defined by the [ReferenceClass](#) entity), if present, defines the strong
1138 type of the reference. The absence of this attribute indicates that this reference is not strongly typed. The

1139 expected behavior is that the REFERENCECLASS attribute shall exist for PROPERTY.REFERENCE
1140 usage in class entities and should not exist for instance entities because the reference class name should
1141 be defined in the property value.

1142 The ClassOrigin and Propagated entities are used in the same manner as for other CIM properties.

5.3.5.7 METHOD

1144 The METHOD element defines a single CIM method. It may have qualifiers, and zero or more
1145 parameters.

1146 The order of the PARAMETER, PARAMETER.REFERENCE, PARAMETER.ARRAY and
1147 PARAMETER.REFARRAY subelements is not significant.

```
1148      <!ELEMENT METHOD
1149          (QUALIFIER*, (PARAMETER | PARAMETER.REFERENCE | PARAMETER.ARRAY |
1150          PARAMETER.REFARRAY)*)>
1151      <!ATTLIST METHOD
1152          %CIMName;
1153          %CIMType;          #IMPLIED
1154          %ClassOrigin;
1155          %Propagated;>
```

1156 The NAME attribute (defined by the CIMName entity) defines the name of the method.

1157 The TYPE attribute (defined by the CIMType entity) defines the method return type, if the method returns
1158 a value. If this attribute is absent, the method shall return no value (that is, it has the special return type
1159 `void`).

5.3.5.8 PARAMETER

1161 The PARAMETER element defines a single (non-array, non-reference) parameter to a CIM method. The
1162 parameter may have zero or more qualifiers.

```
1163      <!ELEMENT PARAMETER (QUALIFIER*)>
1164      <!ATTLIST PARAMETER
1165          %CIMName;
1166          %CIMType;        #REQUIRED>
```

1167 The NAME attribute (defined by the CIMName entity) defines the name of the parameter. The TYPE
1168 attribute (defined by the CIMType entity) defines the CIM type of the parameter.

5.3.5.9 PARAMETER.REFERENCE

1170 The PARAMETER.REFERENCE element defines a single reference parameter to a CIM method. The
1171 parameter may have zero or more qualifiers.

```
1172      <!ELEMENT PARAMETER.REFERENCE (QUALIFIER*)>
1173      <!ATTLIST PARAMETER.REFERENCE
1174          %CIMName;
1175          %ReferenceClass;>
```

1176 The NAME attribute (defined by the CIMName entity) defines the name of the parameter.

1177 The REFERENCECLASS attribute (defined by the ReferenceClass entity), if present, defines the strong
1178 type of the reference. If this attribute is absent, the parameter is assumed to be a reference that is not
1179 strongly typed.

1180 The expected behavior is that the REFERENCECLASS attribute shall exist for
1181 PARAMETER.REFERENCE entities.

5.3.5.10 PARAMETER.ARRAY

1183 The PARAMETER.ARRAY element defines a single parameter to a CIM method that has an array type.
1184 The parameter may have zero or more qualifiers.

```
1185     <!ELEMENT PARAMETER.ARRAY (QUALIFIER*)>
1186     <!ATTLIST PARAMETER.ARRAY
1187        %CIMName;
1188        %CIMType;          #REQUIRED
1189        %ArraySize;>
```

1190 The NAME attribute (defined by the CIMName entity) defines the name of the parameter. The TYPE
1191 attribute (defined by the CIMType entity) defines the CIM type of the parameter.

1192 The ARRAYSIZE attribute (defined by the ArraySize entity) shall be present if the array is a fixed-length
1193 array, and shall be absent if the array is a variable-length array.

5.3.5.11 PARAMETER.REFARRAY

1195 The PARAMETER.REFARRAY element defines a single parameter to a CIM method that has an array of
1196 references type. The parameter may have zero or more qualifiers.

```
1197     <!ELEMENT PARAMETER.REFARRAY (QUALIFIER*)>
1198     <!ATTLIST PARAMETER.REFARRAY
1199        %CIMName;
1200        %ReferenceClass;
1201        %ArraySize;>
```

1202 The NAME attribute (defined by the CIMName entity) defines the name of the parameter.

1203 The REFERENCECLASS attribute (defined by the ReferenceClass entity) defines the strong type of a
1204 reference. If this attribute is absent, the parameter is not a strongly typed reference. The expected
1205 behavior is that the REFERENCECLASS attribute shall exist for PARAMETER.REFARRAY entities.

1206 The ARRAYSIZE attribute (defined by the ArraySize entity) shall be present if the array is a fixed-length
1207 array, and shall be absent if the array is a variable-length array.

5.3.6   Message Elements

1209 This subclause defines those XML elements that are concerned with expressing CIM-XML messages for
1210 DSP0200.

5.3.6.1   MESSAGE

1212 The MESSAGE element models a single CIM-XML message. This element is used as the basis for CIM
1213 Operation Messages and CIM Export Messages.

```
1214     <!ELEMENT MESSAGE
1215        (SIMPLEREQ | MULTIREQ | SIMPLERSP | MULTIRSP |
1216        SIMPLEEXPREQ | MULTIEXPREQ | SIMPLEEXPRSP | MULTIEXPRSP)>
1217     <!ATTLIST MESSAGE
1218        ID CDATA                #REQUIRED
1219        PROTOCOLVERSION CDATA   #REQUIRED>
```

1220 The `ID` attribute defines an identifier for the MESSAGE element. The content of the value is not
1221 constrained by this specification, but the intention is that `ID` attribute be used as a correlation mechanism
1222 between two CIM entities.

1223 The `PROTOCOLVERSION` attribute defines the version of [DSP0200](#) to which this message conforms. It
1224 shall be in the form of "M.N", where M is the major version of the specification in numeric form, and N is
1225 the minor version of the specification in numeric form (for example, 1.0, 1.1). Implementations shall
1226 validate only the major version because all minor versions are backward compatible. Implementations
1227 may look at the minor version to determine additional capabilities.

1228 [DSP0200](#) provides more details on the values that these attributes may take.

5.3.6.2 MULTIREQ

1230 The MULTIREQ element defines a multiple CIM operation request. It contains two or more subelements
1231 that define the [SIMPLEREQ](#) elements that make up this multiple request.

1232   `<!ELEMENT MULTIREQ (SIMPLEREQ, SIMPLEREQ+)>`

5.3.6.3 SIMPLEREQ

1234 The SIMPLEREQ element defines a simple CIM operation request. It contains either a [METHODCALL](#)
1235 (extrinsic method) element or an [IMETHODCALL](#) (intrinsic method) element.

1236 In addition, it contains zero or more [CORRELATOR](#) elements, each representing a client-defined
1237 operation correlator. For details on operation correlators, see [DSP0200](#).

1238   `<!ELEMENT SIMPLEREQ (CORRELATOR*, (METHODCALL | IMETHODCALL))>`

5.3.6.4 METHODCALL

1240 The METHODCALL element defines a single method invocation on a class or instance. It specifies the
1241 local path of the target class or instance, followed by zero or more [PARAMVALUE](#) subelements as the
1242 parameter values to be passed to the method.

1243   `<!ELEMENT METHODCALL ((LOCALCLASSPATH | LOCALINSTANCEPATH), PARAMVALUE*)>`
1244   `<!ATTLIST METHODCALL`
1245   `%CIMName;>`

1246 The NAME attribute (defined by the [CIMName](#) entity) defines the name of the method to be invoked.

5.3.6.5 PARAMVALUE

1248 The PARAMVALUE element defines the value of an input or output parameter of an extrinsic method call,
1249 or the value of an output parameter of an intrinsic method call. Note that input parameters of intrinsic
1250 method calls are represented by the [IPARAMVALUE](#) element; this inconsistency for intrinsic methods has
1251 historical reasons.

1252   `<!ELEMENT PARAMVALUE (`
1253   `VALUE | VALUE.REFERENCE | VALUE.ARRAY | VALUE.REFARRAY |`
1254   `CLASSNAME | INSTANCENAME | CLASS | INSTANCE | VALUE.NAMEDINSTANCE)?>`
1255   `<!ATTLIST PARAMVALUE`
1256   `%CIMName;`
1257   `%ParamType;      #IMPLIED`
1258   `%EmbeddedObject;>`

1259 The child element of the PARAMVALUE element represents the parameter value. The absence of a child
1260 element indicates that the parameter value is NULL.

1261 The NAME attribute (defined by the CIMName entity) defines the name of the parameter.

1262 When PARAMVALUE is used in METHODCALL or METHODRESPONSE (that is, for extrinsic methods),
1263 the following applies:

1264 • The PARAMTYPE attribute (defined by the ParamType entity) if provided identifies the CIM type
1265 of the parameter.
1266 • If the direct child element of the PARAMVALUE element is a VALUE or VALUE.ARRAY
1267 element, the CIM type of the parameter determines the format of the content of the VALUE child
1268 element and CIM-XML producers shall provide the PARAMTYPE attribute with one of the
1269 values defined in Table 1. The requirement to provide the PARAMTYPE attribute was added in
1270 version 2.4.0 of this specification because it supports strongly typed representations of values in
1271 CIM-XML consumers and can be used to improve performance. CIM-XML producers that
1272 support older versions of this specification may not provide the PARAMTYPE attribute for these
1273 direct child elements. In that case, CIM-XML consumers need to have knowledge about the CIM
1274 type in order to interpret the value of (direct or indirect) VALUE child elements correctly (for
1275 example, the uint8 value 3 and the string value "3" are both represented as
1276 `<VALUE>3</VALUE>`). If CIM-XML consumers do not have knowledge about the CIM type, they
1277 should assume as a default that the value is string-typed.
1278 • If the direct child element of the PARAMVALUE element is a VALUE.REFERENCE or
1279 VALUE.REFARRAY element, CIM-XML producers should provide the PARAMTYPE attribute. If
1280 provided, it shall have the value defined in Table 1. Because there is only one possible
1281 PARAMTYPE value for each of these child elements, CIM-XML consumers can infer the CIM
1282 type of the parameter from the name of the direct child element, if the PARAMTYPE attribute is
1283 not provided.
1284 • Other direct child elements of the PARAMVALUE element are not permitted if used in
1285 METHODCALL or METHODRESPONSE.

1286 If the class definition for the extrinsic method parameter has the EMBEDDEDOBJECT or
1287 EMBEDDEDINSTANCE qualifier attached, the EmbeddedObject attribute (defined by the
1288 EmbeddedObject entity) shall be provided, as follows.

1289 • A method parameter that is defined in the class as an embedded object by attaching the
1290 EMBEDDEDOBJECT qualifier on the parameter shall be represented using the
1291 EmbeddedObject attribute with a value of `"object"`. The (string-typed) parameter value shall
1292 be a valid INSTANCE element, defining a single CIM instance of a CIM class or a valid CLASS
1293 element; where these elements shall be escaped as defined in 5.1.1.4.5.
1294 • A method parameter that is defined in the class as an embedded instance by attaching the
1295 EMBEDDEDINSTANCE qualifier on the parameter shall be represented using the
1296 EmbeddedObject attribute with a value of `"instance"`. The (string-typed) parameter value
1297 shall be a valid INSTANCE element, defining a single CIM instance of the CIM class specified in
1298 the EMBEDDEDINSTANCE qualifier; where this element shall be escaped as defined in
1299 5.1.1.4.5.

1300 As a result, if an embedded instance has properties that are again embedded objects, each such level of
1301 embedding will be escaped separately, and thus, recursively.

1302 **Table 1 - Requirements for PARAMVALUE when used in METHODCALL or METHODRESPONSE**

Direct child element of PARAMVALUE	Requirement to provide PARAMTYPE	Allowed PARAMTYPE values
VALUE	shall provide	boolean, string (including for embedded objects and octet strings), char16, uint8, sint8, uint16, sint16, uint32, sint32, uint64, sint64, datetime, real32, real64

Direct child element of PARAMVALUE	Requirement to provide PARAMTYPE	Allowed PARAMTYPE values
VALUE.ARRAY	shall provide	boolean, string (including for embedded objects and octet strings), char16, uint8 (including for octet strings), sint8, uint16, sint16, uint32, sint32, uint64, sint64, datetime, real32, real64
VALUE.REFERENCE	should provide	reference
VALUE.REFARRAY	should provide	reference

1303 When PARAMVALUE is used in IMETHODRESPONSE (that is, for output parameters of intrinsic
1304 methods), the following applies:

1305 • The PARAMTYPE attribute shall not be provided by CIM-XML producers, because the child
1306 element to be used and the datatype of the parameter value is known from the definition of the
1307 intrinsic method. Note that IPARAMVALUE (used for input parameters of intrinsic methods)
1308 does not provide for the specification of PARAMTYPE.
1309 • The CIM types of intrinsic method output parameters and the child elements of PARAMVALUE
1310 that are to be used for representing parameter values are defined in DSP0200.

1311 **5.3.6.6   IMETHODCALL**

1312 The IMETHODCALL element defines a single intrinsic method invocation. It specifies the target local
1313 namespace, followed by zero or more IPARAMVALUE subelements as the parameter values to be
1314 passed to the method.

1315 ```
<!ELEMENT IMETHODCALL (LOCALNAMESPACEPATH, IPARAMVALUE*)>
```
1316 ```
<!ATTLIST IMETHODCALL
```
1317 ```
    %CIMName;>
```

1318 The NAME attribute (defined by the CIMName entity) defines the name of the method to be invoked.

1319 **5.3.6.7   IPARAMVALUE**

1320 The IPARAMVALUE element defines the value of a parameter of an intrinsic method call.

1321 ```
<!ELEMENT IPARAMVALUE
```
1322 ```
    (VALUE | VALUE.ARRAY | VALUE.REFERENCE | CLASSNAME | INSTANCENAME |
```
1323 ```
    QUALIFIER.DECLARATION |CLASS | INSTANCE | VALUE.NAMEDINSTANCE)?>
```
1324 ```
<!ATTLIST IPARAMVALUE
```
1325 ```
    %CIMName;>
```

1326 The child element of the IPARAMVALUE element represents the parameter value. The absence of a child
1327 element indicates that the parameter value is NULL.

1328 The NAME attribute (defined by the CIMName entity) defines the name of the parameter.

1329 The IPARAMVALUE element does not provide information about the CIM type of the parameter. CIM-
1330 XML consumers are expected to have knowledge about the CIM types of intrinsic method parameters.
1331 The CIM types of intrinsic method parameters and the child elements of IPARAMVALUE that are to be
1332 used for representing parameter values are defined in DSP0200.

1333 **5.3.6.8   MULTIRSP**

1334 The MULTIRSP element defines a multiple CIM operation response. It contains two or more subelements
1335 that define the SIMPLERSP elements that make up this multiple response.

1336   `<!ELEMENT MULTIRSP (SIMPLERSP, SIMPLERSP+)>`

5.3.6.9   SIMPLERSP

1337

1338 The SIMPLERSP element defines a simple CIM operation response. It contains either a
1339 METHODRESPONSE (for extrinsic methods) element or an IMETHODRESPONSE (for intrinsic methods)
1340 element.

1341   `<!ELEMENT SIMPLERSP (METHODRESPONSE | IMETHODRESPONSE)>`

5.3.6.10   METHODRESPONSE

1342

1343 The METHODRESPONSE element defines the response to a single CIM extrinsic method invocation. It
1344 contains either an ERROR subelement (to report a fundamental error that prevented the method from
1345 executing) or a combination of an optional return value and zero or more out parameter values.

1346   `<!ELEMENT METHODRESPONSE (ERROR | (RETURNVALUE?, PARAMVALUE*))>`
1347   `<!ATTLIST METHODRESPONSE`
1348   `%CIMName;>`

1349 The NAME attribute (defined by the CIMName entity) defines the name of the method that was invoked.

5.3.6.11   IMETHODRESPONSE

1350

1351 The IMETHODRESPONSE element defines the response to a single intrinsic CIM method invocation. It
1352 contains either an ERROR subelement (to report a fundamental error that prevented the method from
1353 executing) or an optional return value and zero or more out parameter values.

1354   `<!ELEMENT IMETHODRESPONSE (ERROR | (IRETURNVALUE?, PARAMVALUE*))>`
1355   `<!ATTLIST IMETHODRESPONSE`
1356   `%CIMName;>`

1357 The NAME attribute (defined by the CIMName entity) defines the name of the method that was invoked.

5.3.6.12   ERROR

1358

1359 The ERROR element is used to define a fundamental error that prevented a method from executing
1360 normally. It consists of a status code, an optional description, and zero or more instances that contain
1361 detailed information about the error.

1362   `<!ELEMENT ERROR (INSTANCE*)`
1363   `<!ATTLIST ERROR`
1364   `CODE          CDATA  #REQUIRED`
1365   `DESCRIPTION   CDATA  #IMPLIED>`

1366 The CODE attribute contains a numerical status code that indicates the nature of the error. The valid
1367 status codes are defined in DSP0200. The value of the CODE attribute is whitespace-tolerant (see
1368 5.1.1.3).

1369 The DESCRIPTION attribute, if present, provides a human-readable description of the error. The format
1370 of the value of the DESCRIPTION attribute, if provided, shall be a sequence of zero or more UCS
1371 characters and is character-preserving (see 5.1.1.2). The actual representation of UCS characters
1372 depends on the `encoding` attribute defined in the XML declaration (`<?xml ... ?>`).

1373 **5.3.6.13 RETURNVALUE**

1374 The RETURNVALUE element specifies the (scalar) value returned from an extrinsic method call.

```
1375      <!ELEMENT RETURNVALUE (VALUE | VALUE.REFERENCE)?>
1376      <!ATTLIST RETURNVALUE
1377          %EmbeddedObject;
1378          %ParamType;      #IMPLIED>
```

1379 The child element of the RETURNVALUE element represents the returned value. The absence of a child
1380 element indicates that the returned value is NULL.

1381 The PARAMTYPE attribute (defined by the ParamType entity) if provided identifies the CIM type of the
1382 returned value.

1383 If the direct child element of the RETURNVALUE element is a VALUE element, CIM-XML producers shall
1384 provide the PARAMTYPE attribute with one of the values defined in Table 2. The requirement to provide
1385 the PARAMTYPE attribute was added in version 2.4.0 of this specification because it supports strongly
1386 typed representations of values in CIM-XML consumers and can be used to improve performance. CIM-
1387 XML producers that support older versions of this specification may not provide the PARAMTYPE
1388 attribute for these direct child elements. In that case, CIM-XML consumers need to have knowledge about
1389 the CIM type in order to interpret the value of VALUE child elements correctly (for example, the uint8
1390 value 3 and the string value "3" are both represented as `<VALUE>3</VALUE>`). If CIM-XML consumers
1391 do not have knowledge about the CIM type, they should assume as a default that the value is string-
1392 typed.

1393 If the direct child element of the RETURNVALUE element is a VALUE.REFERENCE element, CIM-XML
1394 producers should provide the PARAMTYPE attribute. If provided, it shall have the value defined in Table
1395 2. Because there is only one possible PARAMTYPE value for this child element, CIM-XML consumers
1396 can infer the CIM type of the return value from the name of the direct child element, if the PARAMTYPE
1397 attribute is not provided.

1398 If the class definition for the extrinsic method has the EMBEDDEDOBJECT or EMBEDDEDINSTANCE
1399 qualifier attached, the EmbeddedObject attribute (defined by the EmbeddedObject entity) shall be
1400 provided, as follows.

1401 •   A return value that is defined in the class as an embedded object by attaching the
1402   EMBEDDEDOBJECT qualifier on the method shall be represented using the EmbeddedObject
1403   attribute with a value of `"object"`. The (string-typed) return value shall be a valid INSTANCE
1404   element, defining a single CIM instance of a CIM class or a valid CLASS element; where these
1405   elements shall be escaped as defined in 5.1.1.4.5.
1406 •   A return value that is defined in the class as an embedded instance by attaching the
1407   EMBEDDEDINSTANCE qualifier on the method shall be represented using the
1408   EmbeddedObject attribute with a value of `"instance"`. The (string-typed) return value shall be
1409   a valid INSTANCE element, defining a single CIM instance of the CIM class specified in the
1410   EMBEDDEDINSTANCE qualifier; where this element shall be escaped as defined in 5.1.1.4.5.

1411 As a result, if an embedded instance has properties that are again embedded objects, each such level of
1412 embedding will be escaped separately, and thus, recursively.

1413 **Table 2 - Requirements for RETURNVALUE**

Direct child element of RETURNVALUE	Requirement to provide PARAMTYPE	Allowed PARAMTYPE values
VALUE	shall provide	boolean, string (including for embedded objects and octet strings), char16, uint8, sint8, uint16, sint16, uint32, sint32, uint64, sint64, datetime, real32, real64

Direct child element of RETURNVALUE	Requirement to provide PARAMTYPE	Allowed PARAMTYPE values
VALUE.REFERENCE	should provide	reference

1414

5.3.6.14 IRETURNVALUE

The IRETURNVALUE element specifies the value returned from an intrinsic method call. The absence of a subelement indicates that the return value has the NULL value.

```
<!ELEMENT IRETURNVALUE
    (CLASSNAME* | INSTANCENAME* | VALUE* | VALUE.OBJECTWITHPATH* |
    VALUE.OBJECTWITHLOCALPATH* | VALUE.OBJECT* | OBJECTPATH* |
    QUALIFIER.DECLARATION* | VALUE.ARRAY? | VALUE.REFERENCE? |
    CLASS* | INSTANCE* | INSTANCEPATH* | VALUE.NAMEDINSTANCE* |
    VALUE.INSTANCEWITHPATH)>
```

The child elements of the IRETURNVALUE element represent the returned value. The absence of a child element indicates that the returned value is NULL.

The IRETURNVALUE element does not provide information about the CIM type of the returned value. CIM-XML consumers are expected to have knowledge about the CIM types of intrinsic method return values. The CIM types of intrinsic method return values and the child elements of IRETURNVALUE that are to be used for representing return values are defined in DSP0200.

5.3.6.15 MULTIEXPREQ

The MULTIEXPREQ element defines a multiple CIM export request. It contains two or more subelements that define the SIMPLEEXPREQ elements that make up this multiple request.

```
<!ELEMENT MULTIEXPREQ (SIMPLEEXPREQ, SIMPLEEXPREQ+)>
```

5.3.6.16 SIMPLEEXPREQ

The SIMPLEEXPREQ element defines a simple CIM export request. It contains an EXPMETHODCALL (export method) subelement. In addition, it contains zero or more CORRELATOR elements, each representing a server-defined operation correlator. For details on operation correlators, see DSP0200.

```
<!ELEMENT SIMPLEEXPREQ (CORRELATOR*, EXPMETHODCALL)>
```

5.3.6.17 EXPMETHODCALL

The EXPMETHODCALL element defines a single export method invocation. It specifies zero or more EXPPARAMVALUE subelements as the parameter values to be passed to the method.

```
<!ELEMENT EXPMETHODCALL (EXPPARAMVALUE*)>
<!ATTLIST EXPMETHODCALL
    %CIMName;>
```

The NAME attribute (defined by the CIMName entity) defines the name of the export method to be invoked.

5.3.6.18 MULTIEXPRSP

The MULTIEXPRSP element defines a multiple CIM export response. It contains two or more subelements that define the SIMPLEEXPRSP elements that make up this multiple response.

    <!ELEMENT MULTIEXPRSP (SIMPLEEXPRSP, SIMPLEEXPRSP+)>

5.3.6.19 SIMPLEEXPRSP

The SIMPLEEXPRSP element defines a simple CIM export response. It contains an EXPMETHODRESPONSE (for export methods) subelement.

    <!ELEMENT SIMPLEEXPRSP (EXPMETHODRESPONSE)>

5.3.6.20 EXPMETHODRESPONSE

The EXPMETHODRESPONSE element defines the response to a single export method invocation. It contains either an ERROR subelement (to report a fundamental error that prevented the method from executing) or an optional return value.

    <!ELEMENT EXPMETHODRESPONSE (ERROR | IRETURNVALUE?)>
    <!ATTLIST EXPMETHODRESPONSE
       %CIMName;>

The NAME attribute (defined by the CIMName entity) defines the name of the export method that was invoked.

5.3.6.21 EXPPARAMVALUE

The EXPPARAMVALUE element defines a single export method named parameter value. The absence of a subelement indicates that the parameter has the NULL value.

    <!ELEMENT EXPPARAMVALUE (INSTANCE?)>
    <!ATTLIST EXPPARAMVALUE
       %CIMName;>

The NAME attribute (defined by the CIMName entity) defines the name of the parameter.

5.3.6.22 ENUMERATIONCONTEXT (removed)

In version 2.3, this specification defined an ENUMERATIONCONTEXT element for representing the enumeration context value for pulled enumeration operations. However, that element was not used anywhere and has therefore been removed in version 2.4.0 of this specification. Enumeration context values are now represented like strings, as defined in DSP0200.

5.3.6.23 CORRELATOR

The CORRELATOR element defines a single operation correlator. For a description of the concept of operation correlators, see DSP0200.

    <!ELEMENT CORRELATOR (VALUE)>
    <!ATTLIST CORRELATOR
       %CIMName;
       %CIMType;   #REQUIRED>

The NAME attribute (defined using the CIMName entity) defines the name of the correlator. The correlator name shall conform to the format defined by the following ABNF rule:

1485   `correlator-name = org-id ":" local-id`

1486   `org-id` shall identify the business entity owning the definition of the semantics of the correlator. `org-id`
1487   shall include a copyrighted, trademarked, or otherwise unique name that is owned by that business entity
1488   or that is a registered ID assigned to that business entity by a recognized global authority. In addition, to
1489   ensure uniqueness, `org-id` shall not contain a colon (`:`).

1490   `local-id` shall uniquely identify the correlator within `org-id`.

1491   The TYPE attribute (defined using the [CIMType](#) entity) defines the CIM datatype of the correlator value.

1492   The [VALUE](#) child element defines the value of the correlator.

1493
1494

ANNEX A
(informative)

1495
1496

Change History

Version	Date	Description
2.0.0	1999-06-02	Released as DMTF Final Standard
2.2.0	2007-01-11	Released as DMTF Final Standard
2.3.0	2008-11-11	Released as DMTF Standard
2.3.1	2009-07-29	Released as DMTF Standard
2.4.0	2014-01-16	Released as DMTF Standard, with the following changes: Changes: • Removed ENUMERATIONCONTEXT element because representation of enumeration context value was changed to string in DSP0200 (see 5.3.6.22) • Added requirement to provide the PARAMTYPE attribute of PARAMVALUE element for certain CIM types (see 5.3.6.5) • Added requirement to provide the TYPE attribute of the KEYVALUE element (see 5.3.4.13) • Updated several normative references (see clause 2) Deprecations:: • Deprecated the use of the values "object" and "instance" for the PARAMTYPE attribute as they were not used (see 5.2.10) • Deprecated the VALUETYPE attribute of the KEYVALUE element; use TYPE instead (see 5.3.4.13) Additional functions and requirements: • Added support for operation correlators (see 5.3.6.23) • Added support for, respectively clarified, the representation of special values (NaN, Infinites) for real numbers (see 5.3.3.1.3) Clarifications: • Clarified that PARAMVALUE (and not IPARAMVALUE) is used for output parameters of intrinsic methods in IMETHODRESPONSE (see 5.3.6.5) • Clarified the allowed child elements of PARAMVALUE (see 5.3.6.5) • Removed ordering requirements for DECLGROUP* elements (see 5.3.2.2ff) • Clarified escaping, white space handling, and character repertoire (see 5.1.1) • Clarified XML encoding of embedded instances and objects (see 5.1.1.4.5) • Clarified that CLASSORIGIN indicates the leaf-most class (see 5.2.4) • Clarified precision requirements for real numbers (see 5.3.3.1.3) • Clarified syntax and requirements for HOST element (see 5.3.4.3) • Clarified use of the ARRAYSIZE attribute in any elements, and deprecated its use on the QUALIFIER.DECLARATION element Editorial changes: • Cleaned up terminology • Fixed incorrect normative and bibliographic references • Fixed syntax errors in DTD • Added the missing INSTANCEPATH and VALUE.INSTANCEWITHPATH child element to IRETURNVALUE, in support of the PullInstancePaths, PullInstancesWithPath and Open<XXX>Instances operations.

Bibliography

1498   DMTF DSP0203, *DTD for Representation of CIM in XML 2.4*,
1499   http://www.dmtf.org/standards/published_documents/DSP0203_2.4.dtd

1500   DMTF DSP0207, *WBEM URI Mapping Specification 1.0*,
1501   http://www.dmtf.org/standards/published_documents/DSP0207_1.0.pdf

1502   DMTF DSP8044   *XSD for Representation of CIM in XML 2.4*,
1503   http://schemas.dmtf.org/wbem/cim-xml/2/dsp8044_2.4.xsd

1504   *W3C XML Schema, Part 0: Primer (Second Edition)*, W3C Recommendation, 28 October 2004,
1505   http://www.w3.org/TR/2004/REC-xmlschema-0-20041028

1506   *W3C XML Schema, Part 2: Datatypes (Second Edition)*, W3C Recommendation, 28 October 2004,
1507   http://www.w3.org/TR/2004/REC-xmlschema-2-20041028