



1
2
3
4

Document Identifier: DSP0211

Date: 2015-03-06

Version: 2.0.0

5 **CIM-RS Payload Representation in JSON**

6 **Supersedes: 1.0**

7 **Document Type: Specification**

8 **Document Class: Normative**

9 **Document Status: Published**

10 **Document Language: en-US**

11 Copyright Notice

12 Copyright © 2013–2015 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

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

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27 withdrawn or modified after publication, and shall be indemnified and held harmless by any party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

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

CONTENTS

34	Foreword	5
35	Acknowledgments	5
36	Introduction.....	6
37	Document conventions.....	6
38	Typographical conventions	6
39	ABNF usage conventions	6
40	1 Scope	7
41	2 Normative references	7
42	3 Terms and definitions	8
43	4 Symbols and abbreviated terms.....	10
44	5 Conformance.....	10
45	6 CIM-RS payload representation in JSON	10
46	6.1 Overview	10
47	6.2 Conformance to the JSON grammar	11
48	6.3 Additional encoding requirements	11
49	6.4 Version of the payload representation	11
50	6.5 Internet media type	11
51	6.5.1 Media type fields	11
52	6.5.2 Parameter "version"	12
53	6.5.3 Parameter "typed"	12
54	6.6 Representation of protocol payload elements	13
55	6.6.1 Format of payload element descriptions.....	13
56	6.6.2 Instance payload element.....	15
57	6.6.3 InstanceCollection payload element.....	17
58	6.6.4 Class payload element	18
59	6.6.5 ClassCollection payload element.....	22
60	6.6.6 QualifierType payload element	23
61	6.6.7 QualifierTypeCollection payload element	24
62	6.6.8 MethodRequest payload element	25
63	6.6.9 MethodResponse payload element	26
64	6.6.10 IndicationDeliveryRequest payload element	28
65	6.6.11 ErrorResponse payload element	30
66	6.7 Representation of CIM-RS payload data types in JSON.....	32
67	6.8 Representation of CIM-typed values in JSON	32
68	6.8.1 Representation of CIM real32 and real64 datatypes	35
69	6.8.2 Representation of CIM references.....	36
70	6.8.3 Representation of CIM Null values	36
71	ANNEX A (informative) Change log	37
72	Bibliography	38
73		

74 Tables

75	Table 1 – CIM-RS payload elements	13
76	Table 2 – Placeholders in example syntax description	14
77	Table 3 – Placeholders in syntax descriptions of Instance payload element	16
78	Table 4 – Placeholders in syntax description of InstanceCollection payload element	17
79	Table 5 – Placeholders in syntax description of Class payload element.....	20
80	Table 6 – Placeholders in syntax description of ClassCollection payload element.....	22

81 Table 7 – Placeholders in syntax description of QualifierType payload element 23
82 Table 8 – Placeholders in syntax description of QualifierTypeCollection payload element 24
83 Table 9 – Placeholders in syntax descriptions of MethodRequest payload element..... 25
84 Table 10 – Placeholders in syntax descriptions of MethodResponse payload element..... 27
85 Table 11 – Placeholders in syntax description of IndicationDeliveryRequest payload element..... 29
86 Table 12 – Placeholders in syntax description of ErrorResponse payload element..... 30
87 Table 13 – JSON representations of payload data types 32
88 Table 14 – JSON representations of CIM-typed values 33
89

90

Foreword

91 The *CIM-RS Payload Representation in JSON* (DSP0211) specification was prepared by the DMTF CIM-
92 RS Working Group, based on work of the DMTF CIM-RS Incubator.

93 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
94 management and interoperability. For information about the DMTF, see <http://www.dmtf.org>.

95 Acknowledgments

96 The DMTF acknowledges the following individuals for their contributions to this document:

- 97 • Cornelia Davis, EMC
- 98 • Jim Davis, WS
- 99 • George Ericson, EMC
- 100 • Johannes Holzer, IBM
- 101 • Robert Kieninger, IBM
- 102 • Wojtek Kozaczynski, Microsoft
- 103 • Larry Lamers, VMware
- 104 • Andreas Maier, IBM (editor)
- 105 • Karl Schopmeyer, Inova
- 106 • Bob Tillman, EMC
- 107 • Marvin Waschke, CA Technologies

108

Introduction

109 The information in this document should be sufficient to unambiguously identify the representation of the
110 payload elements defined in [DSP0210](#), in JSON (JavaScript Object Notation).

111 The target audience for this specification is typically implementers who are writing WBEM servers, clients,
112 or listeners supporting the CIM-RS protocol with a payload representation in JSON.

113 Document conventions

114 Typographical conventions

115 The following typographical conventions are used in this document:

- 116 • Document titles are marked in *italics*.
- 117 • ABNF rules and JSON text are in `monospaced font`.

118 ABNF usage conventions

119 Format definitions in this document are specified using ABNF (see [RFC5234](#)), with the following
120 deviations:

- 121 • Literal strings are to be interpreted as case-sensitive UCS characters, as opposed to the
122 definition in [RFC5234](#) that interprets literal strings as case-insensitive US-ASCII characters.

123

CIM-RS Payload Representation in JSON

124 1 Scope

125 This specification is a payload representation specification for the CIM-RS protocol defined in [DSP0210](#),
126 describing a representation of CIM-RS payload elements in JSON (JavaScript Object Notation, see
127 [ECMA-262](#)).

128 Specifically, it describes how the abstract payload elements defined in [DSP0210](#) are represented in
129 JSON and how a JSON representation of these payload elements is identified using an Internet media
130 type.

131 Background information for CIM-RS is described in a white paper, [DSP2032](#).

132 2 Normative references

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

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

139 DMTF DSP0004, *CIM Infrastructure Specification 2.8*,
140 http://www.dmtf.org/standards/published_documents/DSP0004_2.8.pdf

141 DMTF DSP0198, *WBEM Glossary 1.0*,
142 http://www.dmtf.org/standards/published_documents/DSP0198_1.0.pdf

143 DMTF DSP0210, *CIM-RS Protocol 2.0*,
144 http://www.dmtf.org/standards/published_documents/DSP0210_2.0.pdf

145 DMTF DSP0223, *Generic Operations 2.0*,
146 http://www.dmtf.org/standards/published_documents/DSP0223_2.0.pdf

147 IETF RFC5234, *Augmented BNF for Syntax Specifications: ABNF*, January 2008,
148 <http://tools.ietf.org/html/rfc5234>

149 IETF RFC6838, *Media Type Specifications and Registration Procedures*, January 2013,
150 <http://tools.ietf.org/html/rfc6838>

151 IETF RFC6839, *Additional Media Type Structured Syntax Suffixes*, January 2013,
152 <http://tools.ietf.org/html/rfc6839>

153 IETF RFC7159, *The JavaScript Object Notation (JSON) Data Interchange Format*, March 2014,
154 <http://tools.ietf.org/html/rfc7159>

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

157 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards (2004, 5th*
158 *edition)*,
159 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse>

160 The Unicode Consortium, *The Unicode Standard, Version 5.2.0, Annex #15: Unicode Normalization*
161 *Forms*,
162 <http://www.unicode.org/reports/tr15/>

163 3 Terms and definitions

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

166 The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"),
167 "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described
168 in [ISO/IEC Directives, Part 2](#), Annex H. The terms in parenthesis are alternatives for the preceding term,
169 for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that
170 [ISO/IEC Directives, Part 2](#), Annex H specifies additional alternatives. Occurrences of such additional
171 alternatives shall be interpreted in their normal English meaning.

172 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
173 described in [ISO/IEC Directives, Part 2](#), Clause 5.

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

177 The terms defined in [DSP0198](#) and [DSP0210](#) apply to this document. Specifically, this document uses
178 the terms "namespace", "qualifier", "qualifier type", "class", "creation class", "ordinary class",
179 "association", "indication", "instance", "property", "ordinary property", "reference", "method", "parameter",
180 "WBEM client" ("client"), "WBEM server" ("server"), and "WBEM listener" ("listener") defined in [DSP0198](#).

181 This document does not define additional terms; some terms defined in these documents are repeated for
182 convenience.

183 3.1

184 CIM-RS payload data type

185 a data type for CIM-RS payload elements, or components thereof. Also called "payload data type" in this
186 document. Payload data types are abstractly defined in [DSP0210](#), and concretely in CIM-RS payload
187 representation specifications (such as this document), and are thus part of the interface between these
188 documents. For the list of payload data types defined for the CIM-RS protocol, see [DSP0210](#).

189 3.2

190 CIM-RS payload element

191 a particular kind of content of the entity body of the HTTP messages used by the CIM-RS protocol. Also
192 called "payload element" in this document. Payload elements are abstractly defined in [DSP0210](#), and
193 concretely in CIM-RS payload representation specifications (such as this document), and are thus part of
194 the interface between these documents. For the list of payload elements defined for the CIM-RS protocol,
195 see [DSP0210](#).

196 **3.3**197 **CIM-RS payload representation**

198 an encoding format that defines how the abstract payload elements defined in [DSP0210](#) are encoded in
199 the entity body of the HTTP messages used by the CIM-RS protocol. This includes resource
200 representations.

201 **3.4**202 **CIM-RS payload representation specification**

203 a specification that defines a CIM-RS payload representation, such as this document.

204 **3.5**205 **CIM-RS protocol**

206 the RESTful protocol defined in [DSP0210](#), for which this document describes a payload representation in
207 JSON.

208 **3.6**209 **CIM-RS resource**

210 an entity in a WBEM server or WBEM listener that can be referenced using a CIM-RS resource identifier
211 and thus can be the target of an HTTP method in the CIM-RS protocol. Also called "resource" in this
212 document.

213 **3.7**214 **CIM-RS resource identifier**

215 a URI that is a reference to a CIM-RS resource in a WBEM server or WBEM listener, as defined in
216 [DSP0210](#). Also called "resource identifier" in this document.

217 **3.8**218 **Internet media type**

219 a string identification for representation formats in Internet protocols. Originally defined for email
220 attachments and termed "MIME type". Because the CIM-RS protocol is based on HTTP, it uses the
221 definition of media types from section 3.7 of [RFC2616](#).

222 **3.9**223 **Normalization Form C**

224 a normalization form for UCS characters that avoids the use of combining marks where possible and that
225 allows comparing UCS character strings on a per-code-point basis. It is defined in [The Unicode Standard,](#)
226 [Annex #15](#).

227 **3.10**228 **resource representation**

229 a representation of a resource or some aspect thereof, in some format. A particular resource may have
230 any number of representations. The format of a resource representation is identified by a media type. In
231 the CIM-RS protocol, the more general term "payload representation" is used, because not all payload
232 elements are resource representations.

233 **3.11**234 **UCS character**

235 a character from the Universal Character Set defined in [ISO/IEC 10646:2003](#). See [DSP0004](#) for the
236 usage of UCS characters in CIM strings. An alternative term is "Unicode character".

237 **3.12**

238 **UCS code position**

239 a numeric identification for a UCS character in the range of 0x0 to 0x10FFFF, as defined in [ISO/IEC](#)
240 [10646:2003](#).

241 **4 Symbols and abbreviated terms**

242 The abbreviations defined in [DSP0198](#) and [DSP0210](#) apply to this document. Specifically, this document
243 uses the abbreviations "ABNF", "CIM", "IANA", "REST", "UCS", "URI", and "WBEM" defined in [DSP0198](#),
244 and the abbreviations "JSON" and "UTF-8" defined in [DSP0210](#).

245 The following additional abbreviations are used in this document.

246 **4.1**

247 **CIM-RS**

248 **CIM RESTful Services**

249 The RESTful protocol for CIM defined in this document and related documents.

250 **4.2**

251 **ECMAScript**

252 a scripting language that is the standard version of what was called JavaScript. It is defined in [ECMA-](#)
253 [262](#).

254 **5 Conformance**

255 A representation of CIM-RS payload elements in JSON conforms to this document only if it conforms to
256 all normative rules stated in this document.

257 The term "CIM-RS representation in JSON" shall be used only for representations of CIM-RS payload
258 elements in JSON that conform to this document.

259 **6 CIM-RS payload representation in JSON**

260 This clause defines the representation of the CIM-RS payload in JSON.

261 The JSON grammar was originally defined informally in [RFC4627](#), then normatively but in the context of
262 the ECMAScript language in clause 15.12.1 of [ECMA-262](#), then normatively and independently of the
263 ECMAScript language in [ECMA-404](#), and most recently normatively in [RFC7159](#), obsoleting [RFC4627](#).
264 These specifications all agree on the syntactic elements of the JSON syntax, but they differ in some other
265 aspects.

266 This document uses JSON as defined in [RFC7159](#).

267 **6.1 Overview**

268 This subclause describes informally and at a high level how the CIM-RS payload elements defined in
269 [DSP0210](#) are represented in JSON.

270 CIM-RS payload elements are represented as JSON objects. The properties of these JSON objects
271 match the attributes of the payload elements 1:1 in most cases. In some cases, name-related attributes
272 are represented as the names of JSON object members, without repeating them as a property in that
273 JSON object. Nested elements in these payload elements are represented as nested JSON objects.
274 Arrays in these payload elements are represented as JSON arrays. For a normative definition, see 6.6.

275 6.2 Conformance to the JSON grammar

276 The payload representation defined in this document shall conform to the grammar defined by the symbol
277 `JSON-text` defined in [RFC7159](#). This includes the definition of whitespace, character repertoire,
278 character representation, encoding and escaping.

279 Note that [RFC7159](#) provides interoperability guidance for several areas.

280 Version 1 of this document provided for an extended escaping mechanism for Unicode characters outside
281 of the BMP (U+10000 to U+10FFFF) by using a single sequence of "\u" followed by up to six hexadecimal
282 characters. This escaping mechanism is not permitted in [RFC7159](#). In order to support a wide range of
283 existing JSON parsers, version 2 of this document no longer permits this escaping mechanism. [RFC7159](#)
284 defines that Unicode characters outside of the BMP when escaping them have to use surrogate pairs,
285 e.g., U+10102 is represented as "\uD800\uDD02" when escaping it.

286 6.3 Additional encoding requirements

287 The payload representation defined in this document shall be encoded in UTF-8. As a consequence, the
288 character encoding is not being indicated in the CIM-RS payload and does not need to be indicated in the
289 relevant HTTP header fields (`Accept-Charset` and `Content-Charset`).

290 A Unicode byte order mark character (U+FEFF) should not be present in the payload representation.

291 6.4 Version of the payload representation

292 [DSP0210](#) requires that CIM-RS payload representation specifications define a version for the payload
293 representations they define.

294 The full version for the payload representation defined in this document shall be the full version (m.n.u) of
295 this document, without any draft levels.

296 The version of the payload representation is indicated in the `version` parameter of the Internet media
297 type (see 6.5.2).

298 6.5 Internet media type

299 [DSP0210](#) requires that CIM-RS payload representation specifications define a unique Internet media type
300 that identifies the payload representation they define, including its version. This subclause defines that
301 media type, using the fields defined in [RFC6838](#). See [RFC6838](#) for the definition of media types and their
302 components, and [RFC6839](#) for the definition of structured suffixes such as `+json`.

303 6.5.1 Media type fields

304 Type name: `application`

305 Subtype name: `vnd.dmtf.cimrs+json`

306 Required parameters:

- 307 • `version` (see 6.5.2)

308 Optional parameters:

- 309 • `typed` (see 6.5.3)

310 Consumers of this media type shall tolerate and ignore any unknown parameters, for future extensibility.

311 Encoding considerations: 8bit (see section 4.8 of [RFC6838](#))

- 312 Security considerations: See section 12 of [RFC7159](#)
- 313 Interoperability considerations: See [RFC7159](#)
- 314 Fragment identifier considerations: None; this media type has no associated fragment identifiers
- 315 Published specification: This document
- 316 Applications that use this media type: WBEM servers, clients, or listeners supporting the CIM-RS
317 protocol defined in [DSP0210](#), with a payload representation as defined in this document
- 318 Magic number(s): n/a
- 319 File extension(s): n/a
- 320 Macintosh file type code(s): n/a
- 321 Person & email address to contact for further information: DMTF CIM-RS Working Group
322 <cim-rs-wg@dmf.org>
- 323 Intended usage: COMMON
- 324 Restrictions on usage: None; while this media type is intended to be used for implementations of the
325 CIM-RS protocol defined in [DSP0210](#), it may be used elsewhere without restrictions.
- 326 Author: DMTF CIM-RS Working Group <cim-rs-wg@dmf.org>
- 327 Change controller: DMTF CIM-RS Working Group <cim-rs-wg@dmf.org>

328 Example:

```
329 application/vnd.dmtf.cimrs+json; version=2.0.0; typed=true
```

330 6.5.2 Parameter "version"

331 The parameter named "version" is required to be specified, and shall identify the version of the
332 payload representation defined in this document, as described in 6.4, using the following format for its
333 value (defined in ABNF):

```
334 version-value = M "." N [ "." U ]
```

335 where M is the major version indicator, N is the minor version indicator, and U is the update version
336 indicator within the version. Each of these version indicator strings shall be a decimal representation of
337 the corresponding version indicator number without leading zeros. Note that each indicator version string
338 may include more than a single decimal digit.

339 [DSP0210](#) defines additional requirements on the presence of the update version indicator, for uses of this
340 media type in HTTP header fields.

341 6.5.3 Parameter "typed"

342 The parameter named "typed" is optional to be specified, and shall indicate whether the payload
343 representation includes type information for any values, using the following format for its value (defined in
344 ABNF):

```
345 typed-value = "true" / "false"
```

346 If this parameter is not specified in the media type, it shall default to "false".

347 The values of this parameter shall be treated case sensitively.

348 If the value is "true", type information for values shall be included in the payload representation.
 349 Otherwise, type information for values shall not be included.

350 Subclause 6.6 defines for each payload element whether and how exactly it depends on the inclusion of
 351 type information. This creates two variants of some payload elements, dependent on whether type
 352 information is included or not included. These two variants of a payload element may be incompatibly
 353 different. Generally, only representations of instances, method requests and method responses differ
 354 between these variants. However, there is a subtle case where class representations also differ: Default
 355 values of properties that are embedded instances are in fact instance representations and thus differ.

356 Wbem servers when processing server operations and Wbem listeners when processing listener
 357 operations shall support both of those variants.

358 Wbem servers when issuing listener operations and Wbem clients when issuing server operations may
 359 use any of those two variants.

360 6.6 Representation of protocol payload elements

361 This subclause defines how the CIM-RS payload elements defined in [DSP0210](#) are represented in JSON.

362 Table 1 provides an overview of these payload elements and references the subclauses that describe
 363 their representations.

364 **Table 1 – CIM-RS payload elements**

Payload Element	Meaning	Description
Instance	Representation of an instance resource; that is, a modeled object in the managed environment	See 6.6.2
InstanceCollection	A list of representations of instance resources	See 6.6.3
Class	Representation of a class resource; that is, a class declaration	See 6.6.4
ClassCollection	A list of representations of class resources	See 6.6.5
QualifierType	Representation of a qualifier type	See 6.6.6
QualifierTypeCollection	A list of representations of qualifier types	See 6.6.7
MethodRequest	The data describing a method invocation request, including input parameters	See 6.6.8
MethodResponse	The data describing a method invocation response, including its return value and output parameters	See 6.6.9
IndicationDeliveryRequest	The data describing a request to deliver an indication to a listener	See 6.6.10
ErrorResponse	The data describing an error response to any request	See 6.6.11

365 6.6.1 Format of payload element descriptions

366 The following subclauses use a lightweight approach for describing the JSON structure for the various
 367 payload elements.

368 The following example syntax description illustrates this description approach:

```
369 {
370   "kind": "instance",
371   "self": (self),
372   "namespace": (namespace),
373   "classname": (classname),
```

```

374 "properties": {
375     (property-name): {
376         "array": (property-array), ?
377         "arraysize": (property-arraysize), ?
378         "type": (property-type),
379         "classname": (property-classname), ?
380         "value": (property-value)
381     } #
382 } ?
383 }
    
```

384 All text in such a syntax description is to be understood literally as stated, except for whitespace
 385 characters used outside of string literals (see 6.2), and except for the following special indicators:

386 # indicates that the JSON object member or JSON array element to the left of the # may be
 387 present zero or more times in a comma-separated list.

388 ? indicates that the JSON object member or JSON array element to the left of the ? is
 389 optional (that is, it may be present or absent).

390 (self), (namespace), ... are placeholders that correspond to attributes of the represented payload
 391 element. They are meant to be replaced with the JSON representation of the attribute, as
 392 detailed in a table following the syntax description.

393 Table 2 is an example of such a table and shows the placeholders defined in the example syntax
 394 description shown above. The "Attribute of payload element" column shows the attribute corresponding to
 395 the placeholder, in a dotted notation starting with the represented payload element. The "payload data
 396 type" column repeats the payload data type for that attribute. In case of differences between this
 397 repetition and the definition of the payload data type in [DSP0210](#), the definition in [DSP0210](#) is considered
 398 normative and takes precedence. Finally, the "Representation" column references the table or subclause
 399 that defines the representation for that payload data type.

400 **Table 2 – Placeholders in example syntax description**

Placeholder	Attribute of payload element	Payload data type	Representation
(self)	Instance.self	URI	See Table 13
(namespace)	Instance.namespace	String	See Table 13
(classname)	Instance.classname	String	See Table 13
(property-name)	Instance.properties[x].name	String	See Table 13
(property-array)	Instance.properties[x].array	Boolean	See Table 13
(property-arraysize)	Instance.properties[x].arraysize	Integer	See Table 13
(property-type)	Instance.properties[x].type	String	See Table 13
(property-classname)	Instance.properties[x].classname	String	See Table 13
(property-value)	Instance.properties[x].value	Value	See Table 13

401 The use of commas in lists (e.g., in the example above, the list of top level object members, or the list of
 402 name/value pairs that are object members of the "properties" object) is determined by the general JSON
 403 syntax rules; that is, exactly one comma is required between items in a list, and no trailing comma is
 404 permitted after the last list item.


```

450 "classname": (classname),
451 "properties": {
452   (property-name): (property-value)#
453 }?
454 }
    
```

455 If the properties attribute of the represented Instance payload element has no entries, the corresponding
 456 JSON object member ("properties") should not be present (but may be present with a value of an empty
 457 JSON object).

458 See [DSP0210](#) for any other elements marked as optional.

459 Table 3 shows the representation of the placeholders in these syntax descriptions.

Table 3 – Placeholders in syntax descriptions of Instance payload element

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	Instance.self	URI	See Table 13
<i>(namespace)</i>	Instance.namespace	String	See Table 13
<i>(classname)</i>	Instance.classname	String	See Table 13
<i>(property-name)</i>	Instance.properties[x].name	String	See Table 13
<i>(property-array)</i>	Instance.properties[x].array	Boolean	See Table 13
<i>(property-arraysize)</i>	Instance.properties[x].arraysize	Integer	See Table 13
<i>(property-type)</i>	Instance.properties[x].type	String	See Table 13
<i>(property-classname)</i>	Instance.properties[x].classname	String	See Table 13
<i>(property-value)</i>	Instance.properties[x].value	Value	See Table 13

461 Example, if type information is included:

```

462 {
463   "kind": "instance",
464   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11",
465   "namespace": "root/cimv2",
466   "classname": "ACME_VirtualSystem",
467   "properties": {
468     "InstanceID": {
469       "type": "string",
470       "value": "node47:sys11" },
471     "ElementName": {
472       "type": "string",
473       "value": "Virtual system 11 on node 07" },
474     "Caption": {
475       "type": "string",
476       "value": "Virtual system 11 on node 07" }
477   }
478 }
479
    
```


480 Example, if type information is not included:

```

481 {
482   "kind": "instance",
483   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11",
484   "namespace": "root/cimv2",
485   "classname": "ACME_VirtualSystem",
486   "properties": {
487     "InstanceID": "node47:sys11",
488     "ElementName": "Virtual system 11 on node 07",
489     "Caption": "Virtual system 11 on node 07"
490   }
491 }
492 
```

493 **6.6.3 InstanceCollection payload element**

494 InstanceCollection payload elements shall be represented as defined in the following syntax description.
 495 The representation depends on whether type information is included (see 6.5.3), because the
 496 representation of the instances in the collection depends on that:

```

497 {
498   "kind": "instancecollection",
499   "self": (self),
500   "next": (next), ?
501   "instances": [
502     (instance)#
503   ]?
504 }
```

505 If the instances attribute of the represented InstanceCollection payload element has no entries, the
 506 corresponding JSON object member ("instances") should not be present (but may be present with a value
 507 of an empty JSON array).

508 See [DSP0210](#) for any other elements marked as optional.

509 Table 4 shows the representation of the placeholders in this syntax description.

510 **Table 4 – Placeholders in syntax description of InstanceCollection payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	InstanceCollection.self	URI	See Table 13
<i>(next)</i>	InstanceCollection.next	URI	See Table 13
<i>(instance)</i>	InstanceCollection.instances[x]	Instance	See 6.6.2

511 Example, if type information is included:

```

512 {
513   "kind": "instancecollection",
514   "self": "/root%2Fcimv2/classes/ACME_ComputerSystem/instances",
515   "instances": [
516     {
517       "kind": "instance",
```

```

518     "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47
519 %3Asys11",
520     "namespace": "root/cimv2",
521     "classname": "ACME_VirtualSystem",
522     "properties": {
523         "InstanceID": {
524             "type": "string",
525             "value": "node47:sys11" },
526         "ElementName": {
527             "type": "string",
528             "value": "Andy's system" }
529     }
530 },
531     . . . // Other instances
532 ]
533 }

```

534 Example, if type information is not included:

```

535 {
536     "kind": "instancecollection",
537     "self": "/root%2Fcimv2/classes/ACME_ComputerSystem/instances",
538     "instances": [
539         {
540             "kind": "instance",
541             "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47
542 %3Asys11",
543             "namespace": "root/cimv2",
544             "classname": "ACME_VirtualSystem",
545             "properties": {
546                 "InstanceID": "node47:sys11",
547                 "ElementName": "Andy's system"
548             }
549         },
550         . . . // Other instances
551     ]
552 }

```

553 6.6.4 Class payload element

554 Class payload elements and values of the Class payload data type shall be represented as defined in the
555 following syntax description. While the syntax description below is independent of whether type
556 information is included, the actual representation depends on whether type information is included (see
557 6.5.3), because the representation of default values of properties that are embedded instances depends
558 on that:

```

559 {
560     "kind": "class",
561     "self": (self),
562     "namespace": (namespace),
563     "name": (name),
564     "superclassname": (superclassname),

```

```

565 "qualifiers": {
566     (qualifier-name): {
567         "array": (qualifier-array),?
568         "type": (qualifier-type),
569         "value": (qualifier-value)
570     }#
571 },?
572 "properties": {
573     (property-name): {
574         "qualifiers": {
575             (qualifier-name): {
576                 "array": (qualifier-array),?
577                 "type": (qualifier-type),
578                 "value": (qualifier-value)
579             }#
580         },?
581         "array": (property-array),?
582         "arraysize": (property-arraysize),?
583         "type": (property-type),
584         "classname": (property-classname),?
585         "defaultvalue": (property-defaultvalue)? // potential dependency on type
586                                                    // information for embedded inst.
587     }#
588 },?
589 "methods": {
590     (method-name): {
591         "qualifiers": {
592             (qualifier-name): {
593                 "array": (qualifier-array),?
594                 "type": (qualifier-type),
595                 "value": (qualifier-value)
596             }#
597         },?
598         "type": (method-return-type),
599         "classname": (method-return-classname)?
600         "parameters": {
601             (parameter-name): {
602                 "qualifiers": {
603                     (qualifier-name): {
604                         "array": (qualifier-array),?
605                         "type": (qualifier-type),
606                         "value": (qualifier-value)
607                     }#
608                 },?
609                 "array": (parameter-array),?
610                 "arraysize": (parameter-arraysize),?
611                 "type": (parameter-type),
612                 "classname": (parameter-classname)?
613             }#

```

```

614     }?
615     }#
616     }?
617 }
    
```

618 In this syntax description, qualifiers exist at the levels of class, property, method and parameter,
 619 consistent with the CIM architecture. Because the representation of qualifiers at each of these levels is
 620 the same, the syntax diagram shows only one set of placeholders for qualifiers.

621 If the properties, methods, parameters, or any of the qualifiers attributes of the represented Class payload
 622 element has no entries, the corresponding JSON object member should not be present (but may be
 623 present with a value of an empty JSON array).

624 See [DSP0210](#) for any other elements marked as optional.

625 Table 5 shows the representation of the placeholders in this syntax description.

626 **Table 5 – Placeholders in syntax description of Class payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	Class.self	URI	See Table 13
<i>(namespace)</i>	Class.namespace	String	See Table 13
<i>(name)</i>	Class.name	String	See Table 13
<i>(superclassname)</i>	Class.superclassname	String	See Table 13
<i>(qualifier-name)</i>	...qualifiers[x].name	String	See Table 13
<i>(qualifier-array)</i>	...qualifiers[x].array	Boolean	See Table 13
<i>(qualifier-type)</i>	...qualifiers[x].type	String	See Table 13
<i>(qualifier-value)</i>	...qualifiers[x].value	Value	See Table 13
<i>(property-name)</i>	Class.properties[x].name	String	See Table 13
<i>(property-array)</i>	Class.properties[x].array	Boolean	See Table 13
<i>(property-arraysize)</i>	Class.properties[x].arraysize	Integer	See Table 13
<i>(property-type)</i>	Class.properties[x].type	String	See Table 13
<i>(property-classname)</i>	Class.properties[x].classname	String	See Table 13
<i>(property-defaultvalue)</i>	Class.properties[x].defaultvalue	Value	See Table 13
<i>(method-name)</i>	Class.methods[x].name	String	See Table 13
<i>(method-return-type)</i>	Class.methods[x].type	String	See Table 13
<i>(method-return-classname)</i>	Class.methods[x].classname	String	See Table 13
<i>(parameter-name)</i>	Class.parameters[x].name	String	See Table 13
<i>(parameter-array)</i>	Class.parameters[x].array	Boolean	See Table 13
<i>(parameter-arraysize)</i>	Class.parameters[x].arraysize	Integer	See Table 13
<i>(parameter-type)</i>	Class.parameters[x].type	String	See Table 13
<i>(parameter-classname)</i>	Class.parameters[x].classname	String	See Table 13

627 Example (informally using `...` for omissions and `//` for comments):

```
628 {
629     "kind": "class",
630     "self": "/root%2Fcimv2/classes/ACME_VirtualSystem",
631     "namespace": "root/cimv2",
632     "name": "ACME_VirtualSystem",
633     "superclassname": "ACME_ComputerSystem",
634     "qualifiers": {
635         "Description": {
636             // array is omitted
637             "type": "string",
638             "value": "A virtual system.\n . . ."
639         },
640         . . . // Other qualifier values for this class
641     },
642     "properties": {
643         "InstanceID": {
644             "qualifiers" : { . . . },
645             // array and arraysize are omitted
646             "type": "string"
647             // classname is omitted
648             // defaultvalue is omitted
649         },
650         "ElementName": {
651             "qualifiers" : { . . . },
652             // array and arraysize are omitted
653             "type": "string",
654             // classname is omitted
655             "defaultvalue": ""
656         },
657         . . . // Other property definitions for this class
658     },
659     "methods": {
660         "RequestStateChange": {
661             "qualifiers" : { . . . },
662             "type": "uint32"
663             // classname is omitted
664             "parameters": {
665                 "RequestedState": {
666                     "qualifiers" : { . . . },
667                     // array and arraysize are omitted
668                     "type": "uint16"
669                     // classname is omitted
670                 },
671                 . . . // Other parameters of this method
672             }
673         },
674         . . . // Other method definitions for this class
675     }
676 }
```

677 **6.6.5 ClassCollection payload element**

678 ClassCollection payload elements shall be represented as defined in the following syntax descriptions.
 679 While the syntax description below is independent of whether type information is included, the actual
 680 representation depends on whether type information is included (see 6.5.3), because the representation
 681 of default values of properties that are embedded instances depends on that:

```
682 {
683   "kind": "classcollection",
684   "self": (self),
685   "classes": [
686     (class)#
687   ]?
688 }
```

689 If the classes attribute of the represented ClassCollection payload element has no entries, the
 690 corresponding JSON object member ("classes") should not be present (but may be present with a value
 691 of an empty JSON array).

692 Table 6 shows the representation of the placeholders in this syntax description.

693 **Table 6 – Placeholders in syntax description of ClassCollection payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	ClassCollection.self	URI	See Table 13
<i>(class)</i>	ClassCollection.classes[x]	Class	See 6.6.4

694 Example:

```
695 {
696   "kind": "classcollection",
697   "self": "/root%2Fcimv2/classes?$class=ACME_ComputerSystem",
698   "classes": [
699     {
700       "kind": "class",
701       "self": "/root%2Fcimv2/classes/ACME_VirtualSystem",
702       "namespace": "root/cimv2",
703       "name": "ACME_VirtualSystem",
704       "superclassname": "ACME_ComputerSystem",
705       "qualifiers": { . . . },
706       "properties": { . . . }, // potential dependency on type information for
707                               // default value of embedded instance properties
708       "methods": { . . . }
709     },
710     . . . // Other classes
711   ]
712 }
```

713 **6.6.6 QualifierType payload element**

714 QualifierType payload elements and values of the QualifierType payload data type shall be represented
 715 as defined in the following syntax description. The representation does not depend on whether type
 716 information is included:

```

717 {
718   "kind": "qualifiertype",
719   "self": (self),
720   "namespace": (namespace),
721   "name": (name),
722   "array": (array), ?
723   "type": (type),
724   "defaultvalue": (defaultvalue), ?
725   "scopes": [
726     (scope)#
727   ],
728   "propagation": (propagation),
729   "override": (override), ?
730   "translatable": (translatable)?
731 }
```

732 See [DSP0210](#) for any elements marked as optional.

733 Table 7 shows the representation of the placeholders in this syntax description.

734 **Table 7 – Placeholders in syntax description of QualifierType payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	QualifierType.self	URI	See Table 13
<i>(namespace)</i>	QualifierType.namespace	String	See Table 13
<i>(name)</i>	QualifierType.name	String	See Table 13
<i>(array)</i>	QualifierType.array	Boolean	See Table 13
<i>(type)</i>	QualifierType.type	String	See Table 13
<i>(defaultvalue)</i>	QualifierType.defaultvalue	Value	See Table 13
<i>(scope)</i>	QualifierType.scopes[x]	String	See Table 13
<i>(propagation)</i>	QualifierType.propagation	Boolean	See Table 13
<i>(override)</i>	QualifierType.override	Boolean	See Table 13
<i>(translatable)</i>	QualifierType.translatable	Boolean	See Table 13

735 Example:

```

736 {
737   "kind": "qualifiertype",
738   "self": "/root%2Fcimv2/qualifiertypes/Abstract",
739   "namespace": "root/cimv2",
740   "name": "Abstract",
741   // array is omitted
742   "type": "boolean",
```

```

743     "defaultvalue": false,
744     "scopes": ["class", "association", "indication"],
745     "propagation": false,
746     // override is omitted
747     // translatable is omitted
748 }
    
```

749 **6.6.7 QualifierTypeCollection payload element**

750 QualifierTypeCollection payload elements shall be represented as defined in the following syntax
 751 description. The representation does not depend on whether type information is included:

```

752 {
753     "kind": "qualifiertypecollection",
754     "self": (self),
755     "qualifiertypes": [
756         (qualifiertype)#
757     ]?
758 }
    
```

759 If the *qualifiertypes* attribute of the represented QualifierTypeCollection payload element has no entries,
 760 the corresponding JSON object member ("*qualifiertypes*") should not be present (but may be present with
 761 a value of an empty JSON array).

762 Table 8 shows the representation of the placeholders in this syntax description.

763 **Table 8 – Placeholders in syntax description of QualifierTypeCollection payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	QualifierTypeCollection.self	URI	See Table 13
<i>(qualifiertype)</i>	QualifierTypeCollection.qualifiertypes[x]	QualifierType	See 6.6.6

764 Example:

```

765 {
766     "kind": "qualifiertypecollection",
767     "self": "/root%2Fcimv2/qualifiertypes",
768     "qualifiertypes": [
769         {
770             "kind": "qualifiertype",
771             "self": "/root%2Fcimv2/qualifiertypes/Abstract",
772             "namespace": "root/cimv2",
773             "name": "Abstract",
774             // array is omitted
775             "type": "boolean",
776             "defaultvalue": false,
777             "scopes": ["class", "association", "indication"],
778             "propagation": false,
779             // override is omitted
780             // translatable is omitted
781         },
    
```



```

782     . . . // Other qualifier types
783     ]
784 }
    
```

785 **6.6.8 MethodRequest payload element**

786 MethodRequest payload elements shall be represented as defined in the following syntax descriptions.
 787 The representation depends on whether type information is included (see 6.5.3).

788 If type information is included:

```

789 {
790     "kind": "methodrequest",
791     "self": (self),
792     "methodname": (method-name),
793     "parameters": {
794         (parameter-name): {
795             "array": (parameter-array), ?
796             "arraysize": (parameter-arraysize), ?
797             "type": (parameter-type),
798             "classname": (parameter-classname), ?
799             "value": (parameter-value)
800         }#
801     }?
802 }
    
```

803 If type information is not included:

```

804 {
805     "kind": "methodrequest",
806     "self": (self),
807     "methodname": (method-name),
808     "parameters": {
809         (parameter-name): (parameter-value)#
810     }?
811 }
    
```

812 If the parameters attribute of the represented MethodRequest payload element has no entries, the
 813 corresponding JSON object member ("parameters") should not be present (but may be present with a
 814 value of an empty JSON object).

815 See [DSP0210](#) for any other elements marked as optional.

816 Table 9 shows the representation of the placeholders in these syntax descriptions.

817 **Table 9 – Placeholders in syntax descriptions of MethodRequest payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	MethodRequest.self	URI	See Table 13
<i>(method-name)</i>	MethodRequest.methodname	String	See Table 13
<i>(parameter-name)</i>	MethodRequest.parameters[x].name	String	See Table 13
<i>(parameter-array)</i>	MethodRequest.parameters[x].array	Boolean	See Table 13

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(parameter-arraysize)</i>	MethodRequest.parameters[x].arraysize	Integer	See Table 13
<i>(parameter-type)</i>	MethodRequest.parameters[x].type	String	See Table 13
<i>(parameter-classname)</i>	MethodRequest.parameters[x].classname	String	See Table 13
<i>(parameter-value)</i>	MethodRequest.parameters[x].value	Value	See Table 13

818 Example, if type information is included:

```

819 {
820   "kind": " methodrequest",
821   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
822 ys11",
823   "methodname": "RequestStateChange",
824   "parameters": {
825     "RequestedState": {
826       // array and arraysize are omitted
827       "type": "uint16",
828       // classname is omitted
829       "value": 2 },
830     "TimeoutPeriod": {
831       // array and arraysize are omitted
832       "type": "datetime",
833       // classname is omitted
834       "value": null }
835   }
836 }
```

837 Example, if type information is not included:

```

838 {
839   "kind": " methodrequest",
840   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
841 ys11",
842   "methodname": "RequestStateChange",
843   "parameters": {
844     "RequestedState": 2,
845     "TimeoutPeriod": null
846   }
847 }
```

848 6.6.9 MethodResponse payload element

849 MethodResponse payload elements shall be represented as defined in the following syntax descriptions.
850 The representation depends on whether type information is included (see 6.5.3).

851 If type information is included:

```

852 {
853   "kind": "methodresponse",
854   "self": (self),
855   "methodname": (method-name),
```

```

856 "returnValue": {
857   "type": (return-type),
858   "classname": (return-classname),?
859   "value": (return-value)
860 },
861 "parameters": {
862   (parameter-name): {
863     "array": (parameter-array),?
864     "arraysize": (parameter-arraysize),?
865     "type": (parameter-type),
866     "classname": (parameter-classname),?
867     "value": (parameter-value)
868   }#
869 }?
870 }

```

871 If type information is not included:

```

872 {
873   "kind": "methodresponse",
874   "self": (self),
875   "methodname": (method-name),
876   "returnValue": (return-value),
877   "parameters": {
878     (parameter-name): (parameter-value)#
879   }?
880 }

```

881 If the parameters attribute of the represented MethodResponse payload element has no entries, the
 882 corresponding JSON object member ("parameters") should not be present (but may be present with a
 883 value of an empty JSON object).

884 See [DSP0210](#) for any other elements marked as optional.

885 Table 10 shows the representation of the placeholders in these syntax descriptions.

886 **Table 10 – Placeholders in syntax descriptions of MethodResponse payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	MethodResponse.self	URI	See Table 13
<i>(method-name)</i>	MethodResponse.methodname	String	See Table 13
<i>(return-type)</i>	MethodResponse.returnValue.type	String	See Table 13
<i>(return-classname)</i>	MethodResponse.returnValue.classname	String	See Table 13
<i>(return-value)</i>	MethodResponse.returnValue.value	Value	See Table 13
<i>(parameter-name)</i>	MethodResponse.parameters[x].name	String	See Table 13
<i>(parameter-array)</i>	MethodResponse.parameters[x].array	Boolean	See Table 13
<i>(parameter-arraysize)</i>	MethodResponse.parameters[x].arraysize	Integer	See Table 13
<i>(parameter-type)</i>	MethodResponse.parameters[x].type	String	See Table 13

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(parameter-classname)</i>	MethodResponse.parameters[x].classname	String	See Table 13
<i>(parameter-value)</i>	MethodResponse.parameters[x].value	Value	See Table 13

887 Example, if type information is included:

```

888 {
889   "kind": "methodresponse",
890   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
891 ys11",
892   "methodname": "RequestStateChange",
893   "returnvalue": {
894     // array and arraysize are omitted
895     "type": "uint32",
896     "value": 0 },
897   "parameters": {
898     "Job": {
899       // array and arraysize are omitted
900       "type": "reference",
901       "classname": "ACME_Job",
902       "value": null },
903   }
904 }
```

905 Example, if type information is not included:

```

906 {
907   "kind": "methodresponse",
908   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
909 ys11",
910   "methodname": "RequestStateChange",
911   "returnvalue": 0,
912   "parameters": {
913     "Job": null
914   }
915 }
```

916 6.6.10 IndicationDeliveryRequest payload element

917 IndicationDeliveryRequest payload elements shall be represented as defined in the following syntax
918 descriptions. The representation depends on whether type information is included (see 6.5.3), because
919 the representation of the embedded instance depends on that:

```

920 {
921   "kind": "indicationdeliveryrequest",
922   "self": (self),
923   "indication": (indication-instance)
924 }
```

925 Table 11 shows the representation of the placeholders in this syntax description.

926 **Table 11 – Placeholders in syntax description of IndicationDeliveryRequest payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	IndicationDeliveryRequest.self	URI	See Table 13
<i>(indication-instance)</i>	IndicationDeliveryRequest.indication	Instance	See 6.6.2

927 Example, if type information is included:

```

928 {
929   "kind": "indicationdeliveryrequest",
930   "self": "/destinations/dest1/indications",
931   "indication": {
932     "kind": "instance",
933     // self is omitted for embedded instances
934     // namespace is omitted for embedded instances
935     "classname": "ACME_AlertIndication",
936     "properties": {
937       "AlertType": {"type": "uint16", "value": 4},
938       "PerceivedSeverity": {"type": "uint16", "value": 5},
939       "ProbableCause": {"type": "uint16", "value": 42},
940       "Message": {"type": "string",
941                 "value": "BOND0007: Some error happened, rc=23."},
942       "MessageArguments": {"array": true, "type": "string", "value": [ "23" ]},
943       "MessageID": {"type": "string", "value": "BOND0007"},
944       "OwningEntity": {"type": "string", "value": "ACME"}
945     }
946   }
947 }
    
```

948 Example, if type information is not included:

```

949 {
950   "kind": "indicationdeliveryrequest",
951   "self": "/destinations/dest1/indications",
952   "indication": {
953     "kind": "instance",
954     // self is omitted for embedded instances
955     // namespace is omitted for embedded instances
956     "classname": "ACME_AlertIndication",
957     "properties": {
958       "AlertType": 4,
959       "PerceivedSeverity": 5,
960       "ProbableCause": 42,
961       "Message": "BOND0007: Some error happened, rc=23.",
    
```

```

962     "MessageArguments": [ "23" ],
963     "MessageID": "BOND0007",
964     "OwningEntity": "ACME"
965   }
966 }
967 }
```

968 **6.6.11 ErrorResponse payload element**

969 ErrorResponse payload elements shall be represented as defined in the following syntax descriptions.
970 The representation depends on whether type information is included (see 6.5.3), because the
971 representation of the embedded instance depends on that:

```

972 {
973   "kind": "errorresponse",
974   "self": (self),
975   "httpmethod": (httpmethod),
976   "statusCode": (statusCode),
977   "statusdescription": (statusdescription),
978   "errors": [
979     (error-instance)#
980   ]?
981 }
```

982 If the errors attribute of the represented ErrorResponse payload element has no entries, the
983 corresponding JSON object member ("errors") should not be present (but may be present with a value of
984 an empty JSON object).

985 Table 12 shows the representation of the placeholders in this syntax description.

986 **Table 12 – Placeholders in syntax description of ErrorResponse payload element**

Placeholder	Attribute of payload element	Payload data type	Representation
<i>(self)</i>	ErrorResponse.self	URI	See Table 13
<i>(httpmethod)</i>	ErrorResponse.httpmethod	String	See Table 13
<i>(statusCode)</i>	ErrorResponse.statusCode	Integer	See Table 13
<i>(statusdescription)</i>	ErrorResponse.statusdescription	String	See Table 13
<i>(error-instance)</i>	ErrorResponse.errors[x]	Instance	See 6.6.2

987 Example (a failed GET instance), if type information is included:

```

988 {
989   "kind": "errorresponse",
990   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11",
991   "httpmethod": "GET",
992   "statusCode": 6,
993   "statusdescription": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node47:sys11\" does not exist in CIM namespace root/cimv2.",
994   "errors": [
995     {
996   }
```

```

998     "kind": "instance",
999     // self is omitted for embedded instances
1000    // namespace is omitted for embedded instances
1001    "classname": "CIM_Error",
1002    "properties": {
1003      "ErrorType": {
1004        "type": "uint16",
1005        "value": 4},
1006      "ErrorSource": {
1007        "type": "string",
1008        "value": "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""},
1009      "ErrorSourceFormat": {
1010        "type": "uint16",
1011        "value": 2},
1012      "Message": {
1013        "type": "string",
1014        "value": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node47:s
1015 ys11\" does not exist in CIM namespace root/cimv2."},
1016      "MessageArguments": {
1017        "array": true,
1018        "type": "string",
1019        "value": [
1020          "ACME_VirtualSystem.InstanceID=\"node47:sys11\""},
1021          "root/cimv2",
1022          "GetInstance",
1023          null,
1024          "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""
1025        ]},
1026      "MessageID": ": {
1027        "type": "string",
1028        "value": "WIPG0213"},
1029      "OwningEntity": {
1030        "type": "string",
1031        "value": "DMTF"}
1032    }
1033  }
1034 ]
1035 }

```

1036 Example (a failed GET instance), if type information is not included:

```

1037 {
1038   "kind": "errorresponse",
1039   "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
1040 ys11",
1041   "httpmethod": "GET",
1042   "statusCode": 6,
1043   "statusdescription": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node
1044 47:sys11\" does not exist in CIM namespace root/cimv2.",
1045   "errors": [
1046     {

```

```

1047     "kind": "instance",
1048     // self is omitted for embedded instances
1049     // namespace is omitted for embedded instances
1050     "classname": "CIM_Error",
1051     "properties": {
1052         "ErrorType": 4,
1053         "ErrorSource": "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\"",
1054         "ErrorSourceFormat": 2,
1055         "Message": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node47:s
1056 ys11\" does not exist in CIM namespace root/cimv2.",
1057         "MessageArguments": [
1058             "ACME_VirtualSystem.InstanceID=\"node47:sys11\"",
1059             "root/cimv2",
1060             "GetInstance",
1061             null,
1062             "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""
1063         ],
1064         "MessageID": "WIPG0213",
1065         "OwningEntity": "DMTF"
1066     }
1067 }
1068 ]
1069 }

```

1070 **6.7 Representation of CIM-RS payload data types in JSON**

1071 Table 13 lists the JSON data types that shall be used to represent the payload data types that are used in
1072 6.6. Note that because the descriptions in 6.6 resolve any complex payload elements to their leave
1073 elements, Table 13 only needs to show a subset of the payload data types defined in [DSP0210](#).

1074 **Table 13 – JSON representations of payload data types**

Payload data type	JSON data type	Additional rules
Boolean	Boolean	
String	String	See 6.2 for requirements on escaping and encoding
Integer	Number	
URI	String	The string value shall be the CIM-RS resource identifier of the referenced resource in any valid format (see DSP0210)
Value	(varies)	See 6.8

1075 **6.8 Representation of CIM-typed values in JSON**

1076 Table 14 lists the JSON data types that shall be used to represent values that have a CIM data type.

1077

Table 14 – JSON representations of CIM-typed values

CIM data type	JSON data type	Additional rules
boolean	Boolean	Note that JSON is case sensitive w.r.t. the literals <code>true</code> and <code>false</code>
string	String	See 6.2 for requirements on escaping and encoding
char16	String	See 6.2 for requirements on escaping and encoding
string, with OctetString qualifier	String	Shall be represented as if it was a normal CIM string-typed value
uint8[], with OctetString qualifier	Number array	Shall be represented as if it was a normal uint8-array-typed value
string, with EmbeddedInstance qualifier	Object	The embedded instance shall be represented as a JSON object as defined in 6.6.2. Its <i>class</i> attribute shall be the name of the creation class of the embedded instance. Note that the creation class may differ from the class specified in the value of the EmbeddedInstance qualifier.
string, with EmbeddedObject qualifier containing an embedded instance	Object	The embedded instance shall be represented as a JSON object as defined in 6.6.2.
string, with EmbeddedObject qualifier containing an embedded class	Object	The embedded class shall be represented as a JSON object as defined in 6.6.4.
datetime	String	The string value shall be the 25-character datetime string defined in DSP0004
uint8,16,32,64	Number	
sint8,16,32,64	Number	
real32,64	Number or String	See 6.8.1
<classname> ref	String	See 6.8.2
string, with Reference qualifier	String	See 6.8.2
array of any CIM type	array of corresponding JSON type	The type string shall reflect the type of a single array entry

1078 Examples for representing named CIM elements (that is, properties or parameters) of these data types:

```

1079 . . .
1080   "ABoolean": {
1081     "type": "boolean",
1082     "value": true},
1083   "AString": {
1084     "type": "string",
1085     "value": "some text"},
1086   "AChar16": {
1087     "type": "char16",
1088     "value": "Z"},

```

```
1089     "AnOctetstringViaString": {
1090         "type": "string",
1091         "value": "0x00000007616263"},
1092     "AnOctetstringViaUint8Array": {
1093         "array": true,
1094         "type": "uint8",
1095         "value": [ 0, 0, 0, 7, 0x61, 0x62, 0x63 ]},
1096     "AnEmbeddedInstance": {
1097         "type": "instance",
1098         "value": . . . // Instance payload element, see 6.6.2
1099     },
1100     "AnEmbeddedObjectThatIsAnInstance": {
1101         "type": "instance",
1102         "value": . . . // Instance payload element, see 6.6.2
1103     },
1104     "AnEmbeddedObjectThatIsAClass": {
1105         "type": "class",
1106         "value": . . . // Class payload element, see 6.6.4
1107     },
1108     "ADatetime": {
1109         "type": "datetime",
1110         "value": "20120213175830.123456+060"},
1111     "AUint16": {
1112         "type": "uint16",
1113         "value": 20000},
1114     "ASint16": {
1115         "type": "sint16",
1116         "value": -16000},
1117     "AReal32": {
1118         "type": "real32",
1119         "value": 3.1415927},
1120     "ARef": {
1121         "type": "reference",
1122         "classname": "CIM_ComputerSystem",
1123         "value": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%
1124 3Asys11"},
1125     "ABooleanArray": {
1126         "array": true,
1127         "type": "boolean",
1128         "value": [ true, false, null ]},
1129     "AFixedSizeBooleanArray": {
1130         "array": true,
1131         "arraysize": 3,
1132         "type": "boolean",
1133         "value": [ true, false, null ]},
1134     "AStringArray": {
1135         "array": true,
1136         "type": "string",
1137         "value": [ "some text", null, "more text\n" ]},
```

```

1138 "AReal64Array": {
1139     "array": true,
1140     "type": "real64",
1141     "value": [ 1E-42, "NaN", null, "-Infinity" ]},
1142 "ARefArray": {
1143     "array": true,
1144     "type": "reference",
1145     "classname": "CIM_ComputerSystem",
1146     "value": [
1147         "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11
1148     ",
1149         null,
1150         "/root%2Fcimv2/classes/ACME_ComputerSystem/instances/InstanceID=node7"
1151     ]},
1152     . . .

```

1153 6.8.1 Representation of CIM real32 and real64 datatypes

1154 The CIM real32 and real64 types are based on the [IEEE 754](#) Single and Double formats (see [DSP0004](#));
 1155 values of these types shall be represented in JSON as follows, depending on their value:

- 1156 • the [IEEE 754](#) special values *positive infinity*, *negative infinity*, and any *not-a-number* values
 1157 shall be represented as JSON String-typed values using the following strings:

1158 positive infinity: "Infinity"

1159 negative infinity: "-Infinity"

1160 any not-a-number values: "NaN"

1161 NOTE These strings are consistent with Python's serialization of float-typed values in JSON, and with
 1162 Java's serialization of float-typed values as strings. These strings are not consistent with the
 1163 representation of the XML datatypes `xs:float` and `xs:double`.

- 1164 • any other values are normal floating point numbers and shall be represented as JSON Number-
 1165 typed values, using a precision for the significand of at least 9 decimal digits for real32 and at
 1166 least 17 digits for real64. Trailing 0's after the decimal point in the significand may be omitted.

1167 NOTE JSON numbers only support lexical notations with a basis of 10 (e.g., 4.56E-3). The value space
 1168 of CIM real32- and real64-typed values is defined by the [IEEE 754](#) Single and Double formats, which have
 1169 a basis of 2. The definition of a minimum precision for the significand guarantees that the value of CIM real
 1170 types does not change when converting it back and forth between the (10-based) JSON representation
 1171 and a (2-based) internal representation. For details, see subclause 5.6 in [IEEE 754](#).

1172 Examples:

```

1173     . . .
1174     "Throughput": {
1175         "type": "real32",
1176         "value": 3.45E3},
1177     "ErrorRate": {
1178         "type": "real32",
1179         "value": "NaN"},
1180     "LowerLimit": {

```

```
1181     "type": "real64",
1182     "value": "-Infinity"},
1183     . . .
```

1184 6.8.2 Representation of CIM references

1185 Values of CIM reference-typed elements (that is, declared with type <classname> ref) and of reference-
1186 qualified elements (that is, declared with string type and the Reference qualifier set to <classname>) shall
1187 be represented in JSON as a JSON Object such that the JSON value is the CIM-RS resource identifier of
1188 the referenced instance in any valid format defined in [DSP0210](#).

1189 Example:

```
1190     . . .
1191     "System": {
1192         "type": "reference",
1193         "classname": "CIM_ComputerSystem",
1194         "value": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%
1195         3Asys11"}
1196     . . .
```

1197 6.8.3 Representation of CIM Null values

1198 CIM Null values shall be represented using the JSON literal `null`.

1199 Note that the JSON literal `null` is case sensitive.

1200 Example:

```
1201     . . .
1202     "ElementName": { "type": "string", "value": null},
1203     "PossibleStates": { "type": "uint16", "array": true, "value": [1, null, 3]},
1204     . . .
```

ANNEX A
(informative)**Change log**

Version	Date	Description
1.0.0	2013-01-24	
1.0.1	2014-02-11	
2.0.0	2015-03-06	Released as a DMTF Standard, with the following changes: <ul style="list-style-type: none">• Added representation of data type• Upgraded to version 2.0 of generic operations (DSP0223)• Several other changes

Bibliography

1210

1211 This bibliography contains a list of non-normative references for this document.

1212 DMTF DSP2032, *CIM-RS White Paper 2.0*,

1213 http://www.dmtf.org/standards/published_documents/DSP2032_2.0.pdf

1214 ECMA-262, *ECMAScript Language Specification, Edition 5.1*, June 2011,

1215 <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

1216 ECMA-404, *The JSON Data Interchange Format, 1st Edition*, October 2013,

1217 <http://www.ecma-international.org/publications/standards/Ecma-404.htm>

1218 IANA MIME Media Types,

1219 <http://www.iana.org/assignments/media-types/>

1220 IETF RFC4627 (Informational), *The application/json Media Type for JavaScript Object Notation (JSON)*,

1221 July 2006,

1222 <http://tools.ietf.org/html/rfc4627>

1223 J. Holzer, *RESTful Web Services and JSON for WBEM Operations*, Master thesis, University of Applied

1224 Sciences, Konstanz, Germany, June 2009,

1225 <http://mond.htwg-konstanz.de/Abschlussarbeiten/Details.aspx?id=1120>