Document Identifier: DSP0211

Date: 2015-03-06

Version: 2.0.0

CIM-RS Payload Representation in JSON

Supersedes: 1.0

Document Type: Specification

Document Class: Normative

Document Status: Published

Document Language: en-US

CONTENTS

Tables

Foreword

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

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

Acknowledgments

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

- Cornelia Davis, EMC
- Jim Davis, WS
- George Ericson, EMC
- Johannes Holzer, IBM
- Robert Kieninger, IBM
- Wojtek Kozaczynski, Microsoft
- Larry Lamers, VMware
- Andreas Maier, IBM (editor)
- Karl Schopmeyer, Inova
- Bob Tillman, EMC
- Marvin Waschke, CA Technologies

Introduction

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

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

Document conventions

Typographical conventions

The following typographical conventions are used in this document:

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

ABNF usage conventions

Format definitions in this document are specified using ABNF (see RFC5234), with the following deviations:

- Literal strings are to be interpreted as case-sensitive UCS characters, as opposed to the 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

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

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

3 Terms and definitions

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

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

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

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

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

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

3.1
CIM-RS payload data type

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

3.2
CIM-RS payload element

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

3.3

CIM-RS payload representation

an encoding format that defines how the abstract payload elements defined in DSP0210 are encoded in the entity body of the HTTP messages used by the CIM-RS protocol. This includes resource representations.

3.4

CIM-RS payload representation specification

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

3.5

CIM-RS protocol

the RESTful protocol defined in DSP0210, for which this document describes a payload representation in JSON.

3.6

CIM-RS resource

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

3.7

CIM-RS resource identifier

a URI that is a reference to a CIM-RS resource in a WBEM server or WBEM listener, as defined in DSP0210. Also called "resource identifier" in this document.

3.8

Internet media type

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

3.9

Normalization Form C

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

3.10

resource representation

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

3.11

UCS character

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

3.12
UCS code position

a numeric identification for a UCS character in the range of 0x0 to 0x10FFFF, as defined in ISO/IEC 10646:2003.

4 Symbols and abbreviated terms

The abbreviations defined in DSP0198 and DSP0210 apply to this document. Specifically, this document uses the abbreviations "ABNF", "CIM", "IANA", "REST", "UCS", "URI", and "WBEM" defined in DSP0198, and the abbreviations "JSON" and "UTF-8" defined in DSP0210.

The following additional abbreviations are used in this document.

4.1
CIM-RS
CIM RESTful Services

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

4.2
ECMAScript

a scripting language that is the standard version of what was called JavaScript. It is defined in ECMA-262.

5 Conformance

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

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

6 CIM-RS payload representation in JSON

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

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

This document uses JSON as defined in RFC7159.

6.1 Overview

This subclause describes informally and at a high level how the CIM-RS payload elements defined in DSP0210 are represented in JSON.

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

6.2 Conformance to the JSON grammar

The payload representation defined in this document shall conform to the grammar defined by the symbol `JSON-text` defined in RFC7159. This includes the definition of whitespace, character repertoire, character representation, encoding and escaping.

Note that RFC7159 provides interoperability guidance for several areas.

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

6.3 Additional encoding requirements

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

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

6.4 Version of the payload representation

DSP0210 requires that CIM-RS payload representation specifications define a version for the payload representations they define.

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

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

6.5 Internet media type

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

6.5.1 Media type fields

Type name: `application`

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

Required parameters:

- `version` (see 6.5.2)

Optional parameters:

- `typed` (see 6.5.3)

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

Encoding considerations: 8bit (see section 4.8 of RFC6838)

Security considerations: See section 12 of RFC7159

Interoperability considerations: See RFC7159

Fragment identifier considerations: None; this media type has no associated fragment identifiers

Published specification: This document

Applications that use this media type: WBEM servers, clients, or listeners supporting the CIM-RS protocol defined in DSP0210, with a payload representation as defined in this document

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person & email address to contact for further information: DMTF CIM-RS Working Group <cim-rs-wg@dmtf.org>

Intended usage: COMMON

Restrictions on usage: None; while this media type is intended to be used for implementations of the CIM-RS protocol defined in DSP0210, it may be used elsewhere without restrictions.

Author: DMTF CIM-RS Working Group <cim-rs-wg@dmtf.org>

Change controller: DMTF CIM-RS Working Group <cim-rs-wg@dmtf.org>

Example:

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

6.5.2 Parameter "version"

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

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

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

DSP0210 defines additional requirements on the presence of the update version indicator, for uses of this media type in HTTP header fields.

6.5.3 Parameter "typed"

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

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

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

The values of this parameter shall be treated case sensitively.

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

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

WBEM servers when processing server operations and WBEM listeners when processing listener operations shall support both of those variants.

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

6.6 Representation of protocol payload elements

This subclause defines how the CIM-RS payload elements defined in DSP0210 are represented in JSON.

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

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

6.6.1 Format of payload element descriptions

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

The following example syntax description illustrates this description approach:

```
{
  "kind": "instance",
  "self": (self),
  "namespace": (namespace),
  "classname": (classname),
```

```
  "properties": {
    (property-name): {
      "array": (property-array),?
      "arraysize": (property-arraysize),?
      "type": (property-type),
      "classname": (property-classname),?
      "value": (property-value)
    }#
  }?
}
```

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

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

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

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

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

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

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

An example for a valid payload element conforming to the example syntax description shown above would be:

```
{
  "kind": "instance",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11",
  "namespace": "root/cimv2",
  "classname": "ACME_VirtualSystem",
  "properties": {
    "InstanceID": {
      "type": "string",
      "value": "node47:sys11" },
    "ElementName": {
      "type": "string",
      "value": "Virtual system 11 on node 07" },
    "Caption": {
      "type": "string",
      "value": "Virtual system 11 on node 07" }
  }
}
```

6.6.2 Instance payload element

Instance payload elements and values of the Instance payload data type shall be represented as defined in the following syntax descriptions. The representation depends on whether type information is included (see 6.5.3).

If type information is included:

```
{
  "kind": "instance",
  "self": (self),?
  "namespace": (namespace),?
  "classname": (classname),
  "properties": {
    (property-name): {
      "array": (property-array),?
      "arraysize": (property-arraysize),?
      "type": (property-type),
      "classname": (property-classname),?
      "value": (property-value)
    }#
  }?
}
```

If type information is not included:

```
{
  "kind": "instance",
  "self": (self),?
  "namespace": (namespace),?
```

```
  "classname": (classname),
  "properties": {
    (property-name): (property-value)#
  }?
}
```

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

See DSP0210 for any other elements marked as optional.

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
(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

Example, if type information is included:

```
{
  "kind": "instance",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "namespace": "root/cimv2",
  "classname": "ACME_VirtualSystem",
  "properties": {
    "InstanceID": {
      "type": "string",
      "value": "node47:sys11" },
    "ElementName": {
      "type": "string",
      "value": "Virtual system 11 on node 07" },
    "Caption": {
      "type": "string",
      "value": "Virtual system 11 on node 07" }
  }
}
```

Example, if type information is not included:

```
{
  "kind": "instance",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11",
  "namespace": "root/cimv2",
  "classname": "ACME_VirtualSystem",
  "properties": {
    "InstanceID": "node47:sys11",
    "ElementName": "Virtual system 11 on node 07",
    "Caption": "Virtual system 11 on node 07"
  }
}
```

6.6.3 InstanceCollection payload element

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

```
{
  "kind": "instancecollection",
  "self": (self),
  "next": (next),?
  "instances": [
    (instance)#
  ]?
}
```

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

See DSP0210 for any other elements marked as optional.

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

Table 4 – Placeholders in syntax description of InstanceCollection payload element

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

Example, if type information is included:

```
{
  "kind": "instancecollection",
  "self": "/root%2Fcimv2/classes/ACME_ComputerSystem/instances",
  "instances": [
    {
      "kind": "instance",
```

```
      "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47
%3Asys11",
      "namespace": "root/cimv2",
      "classname": "ACME_VirtualSystem",
      "properties": {
        "InstanceID": {
          "type": "string",
          "value": "node47:sys11" },
        "ElementName": {
          "type": "string",
          "value": "Andy's system" }
      }
    },
    . . .  // Other instances
  ]
}
```

Example, if type information is not included:

```
{
  "kind": "instancecollection",
  "self": "/root%2Fcimv2/classes/ACME_ComputerSystem/instances",
  "instances": [
    {
      "kind": "instance",
      "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47
%3Asys11",
      "namespace": "root/cimv2",
      "classname": "ACME_VirtualSystem",
      "properties": {
        "InstanceID": "node47:sys11",
        "ElementName": "Andy's system"
      }
    },
    . . .  // Other instances
  ]
}
```

6.6.4 Class payload element

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

```
{
  "kind": "class",
  "self": (self),
  "namespace": (namespace),
  "name": (name),
  "superclassname": (superclassname),
```

```
"qualifiers": {
  (qualifier-name): {
    "array": (qualifier-array),?
    "type": (qualifier-type),
    "value": (qualifier-value)
  }#
},?
"properties": {
  (property-name): {
    "qualifiers": {
      (qualifier-name): {
        "array": (qualifier-array),?
        "type": (qualifier-type),
        "value": (qualifier-value)
      }#
    },?
    "array": (property-array),?
    "arraysize": (property-arraysize),?
    "type": (property-type),
    "classname": (property-classname),?
    "defaultvalue": (property-defaultvalue)? // potential dependency on type
                                             // information for embedded inst.
  }#
},?
"methods": {
  (method-name): {
    "qualifiers": {
      (qualifier-name): {
        "array": (qualifier-array),?
        "type": (qualifier-type),
        "value": (qualifier-value)
      }#
    },?
    "type": (method-return-type),
    "classname": (method-return-classname)?
    "parameters": {
      (parameter-name): {
        "qualifiers": {
          (qualifier-name): {
            "array": (qualifier-array),?
            "type": (qualifier-type),
            "value": (qualifier-value)
          }#
        },?
        "array": (parameter-array),?
        "arraysize": (parameter-arraysize),?
        "type": (parameter-type),
        "classname": (parameter-classname)?
      }#
```

```
        }?
      }#
    }?
}
```

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

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

See DSP0210 for any other elements marked as optional.

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

Table 5 – Placeholders in syntax description of Class payload element

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

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

```
{
  "kind": "class",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem",
  "namespace": "root/cimv2",
  "name": "ACME_VirtualSystem",
  "superclassname": "ACME_ComputerSystem",
  "qualifiers": {
    "Description": {
      // array is omitted
      "type": "string",
      "value": "A virtual system.\n . . ."
    },
    . . .  // Other qualifier values for this class
  },
  "properties": {
    "InstanceID": {
      "qualifiers" : { . . . },
      // array and arraysize are omitted
      "type": "string"
      // classname is omitted
      // defaultvalue is omitted
    },
    "ElementName": {
      "qualifiers" : { . . . },
      // array and arraysize are omitted
      "type": "string",
      // classname is omitted
      "defaultvalue": ""
    },
    . . .  // Other property definitions for this class
  },
  "methods": {
    "RequestStateChange": {
      "qualifiers" : { . . . },
      "type": "uint32"
      // classname is omitted
      "parameters": {
        "RequestedState": {
          "qualifiers" : { . . . },
          // array and arraysize are omitted
          "type": "uint16"
          // classname is omitted
        },
        . . .  // Other parameters of this method
      }
    },
    . . .  // Other method definitions for this class
  }
}
```

6.6.5 ClassCollection payload element

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

```
{
  "kind": "classcollection",
  "self": (self),
  "classes": [
    (class)#
  ]?
}
```

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

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

Table 6 – Placeholders in syntax description of ClassCollection payload element

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

Example:

```
{
  "kind": "classcollection",
  "self": "/root%2Fcimv2/classes?$class=ACME_ComputerSystem",
  "classes": [
    {
      "kind": "class",
      "self": "/root%2Fcimv2/classes/ACME_VirtualSystem",
      "namespace": "root/cimv2",
      "name": "ACME_VirtualSystem",
      "superclassname": "ACME_ComputerSystem",
      "qualifiers": { . . . },
      "properties": { . . . }, // potential dependency on type information for
                               // default value of embedded instance properties
      "methods": { . . . }
    },
    . . .  // Other classes
  ]
}
```

6.6.6 QualifierType payload element

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

```
{
  "kind": "qualifiertype",
  "self": (self),
  "namespace": (namespace),
  "name": (name),
  "array": (array),?
  "type": (type),
  "defaultvalue": (defaultvalue),?
  "scopes": [
    (scope)#
  ],
  "propagation": (propagation),
  "override": (override),?
  "translatable": (translatable)?
}
```

See DSP0210 for any elements marked as optional.

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

Table 7 – Placeholders in syntax description of QualifierType payload element

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

Example:

```
{
  "kind": "qualifiertype",
  "self": "/root%2Fcimv2/qualifiertypes/Abstract",
  "namespace": "root/cimv2",
  "name": "Abstract",
  // array is omitted
  "type": "boolean",
```

```
  "defaultvalue": false,
  "scopes": ["class", "association", "indication"],
  "propagation": false,
  // override is omitted
  // translatable is omitted
}
```

6.6.7 QualifierTypeCollection payload element

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

```
{
  "kind": "qualifiertypecollection",
  "self": (self),
  "qualifiertypes": [
    (qualifiertype)#
  ]?
}
```

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

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

Table 8 – Placeholders in syntax description of QualifierTypeCollection payload element

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

Example:

```
{
  "kind": "qualifiertypecollection",
  "self": "/root%2Fcimv2/qualifiertypes",
  "qualifiertypes": [
    {
      "kind": "qualifiertype",
      "self": "/root%2Fcimv2/qualifiertypes/Abstract",
      "namespace": "root/cimv2",
      "name": "Abstract",
      // array is omitted
      "type": "boolean",
      "defaultvalue": false,
      "scopes": ["class", "association", "indication"],
      "propagation": false,
      // override is omitted
      // translatable is omitted
    },
```

```
    . . .  // Other qualifier types
  ]
}
```

6.6.8 MethodRequest payload element

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

If type information is included:

```
{
  "kind": "methodrequest",
  "self": (self),
  "methodname": (method-name),
  "parameters": {
    (parameter-name): {
      "array": (parameter-array),?
      "arraysize": (parameter-arraysize),?
      "type": (parameter-type),
      "classname": (parameter-classname),?
      "value": (parameter-value)
    }#
  }?
}
```

If type information is not included:

```
{
  "kind": "methodrequest",
  "self": (self),
  "methodname": (method-name),
  "parameters": {
    (parameter-name): (parameter-value)#
  }?
}
```

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

See DSP0210 for any other elements marked as optional.

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

Table 9 – Placeholders in syntax descriptions of MethodRequest payload element

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

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

Example, if type information is included:

```
{
  "kind": " methodrequest",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "methodname": "RequestStateChange",
  "parameters": {
    "RequestedState": {
      // array and arraysize are omitted
      "type": "uint16",
      // classname is omitted
      "value": 2 },
    "TimeoutPeriod": {
      // array and arraysize are omitted
      "type": "datetime",
      // classname is omitted
      "value": null }
  }
}
```

Example, if type information is not included:

```
{
  "kind": " methodrequest",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "methodname": "RequestStateChange",
  "parameters": {
    "RequestedState": 2,
    "TimeoutPeriod": null
  }
}
```

6.6.9 MethodResponse payload element

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

If type information is included:

```
{
  "kind": "methodresponse",
  "self": (self),
  "methodname": (method-name),
```

```
  "returnvalue": {
    "type": (return-type),
    "classname": (return-classname),?
    "value": (return-value)
  },
  "parameters": {
    (parameter-name): {
      "array": (parameter-array),?
      "arraysize": (parameter-arraysize),?
      "type": (parameter-type),
      "classname": (parameter-classname),?
      "value": (parameter-value)
    }#
  }?
}
```

If type information is not included:

```
{
  "kind": "methodresponse",
  "self": (self),
  "methodname": (method-name),
  "returnvalue": (return-value),
  "parameters": {
    (parameter-name): (parameter-value)#
  }?
}
```

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

See DSP0210 for any other elements marked as optional.

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

Table 10 – Placeholders in syntax descriptions of MethodResponse payload element

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

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

Example, if type information is included:

```
{
  "kind": "methodresponse",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "methodname": "RequestStateChange",
  "returnvalue": {
    // array and arraysize are omitted
    "type": "uint32",
    "value": 0 },
  "parameters": {
    "Job": {
      // array and arraysize are omitted
      "type": "reference",
      "classname": "ACME_Job",
      "value": null },
  }
}
```

Example, if type information is not included:

```
{
  "kind": "methodresponse",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "methodname": "RequestStateChange",
  "returnvalue": 0,
  "parameters": {
    "Job": null
  }
}
```

6.6.10 IndicationDeliveryRequest payload element

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

```
{
  "kind": "indicationdeliveryrequest",
  "self": (self),
  "indication": (indication-instance)
}
```

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
(self)	IndicationDeliveryRequest.self	URI	See Table 13
(indication-instance)	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.",
```

```
      "MessageArguments": [ "23" ],
      "MessageID": "BOND0007",
      "OwningEntity": "ACME"
    }
  }
}
```

6.6.11 ErrorResponse payload element

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

```
{
  "kind": "errorresponse",
  "self": (self),
  "httpmethod": (httpmethod),
  "statuscode": (statuscode),
  "statusdescription": (statusdescription),
  "errors": [
    (error-instance)#
  ]?
}
```

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

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

Table 12 – Placeholders in syntax description of ErrorResponse payload element

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

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

```
{
  "kind": "errorresponse",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "httpmethod": "GET",
  "statuscode": 6,
  "statusdescription": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node
47:sys11\" does not exist in CIM namespace root/cimv2.",
  "errors": [
    {
```

```
      "kind": "instance",
      // self is omitted for embedded instances
      // namespace is omitted for embedded instances
      "classname": "CIM_Error",
      "properties": {
        "ErrorType": {
          "type": "uint16",
          "value": 4},
        "ErrorSource": {
          "type": "string",
          "value": "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""},
        "ErrorSourceFormat": {
          "type": "uint16",
          "value": 2},
        "Message": {
          "type": "string",
          "value": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node47:s
ys11\" does not exist in CIM namespace root/cimv2."},
        "MessageArguments": {
          "array": true,
          "type": "string",
          "value": [
            "ACME_VirtualSystem.InstanceID=\"node47:sys11\"",
            "root/cimv2",
            "GetInstance",
            null,
            "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""
        ]},
        "MessageID": ": {
          "type": "string",
          "value": "WIPG0213"},
        "OwningEntity": {
          "type": "string",
          "value": "DMTF"}
      }
    }
  ]
}
```

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

```
{
  "kind": "errorresponse",
  "self": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3As
ys11",
  "httpmethod": "GET",
  "statuscode": 6,
  "statusdescription": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node
47:sys11\" does not exist in CIM namespace root/cimv2.",
  "errors": [
    {
```

```
      "kind": "instance",
      // self is omitted for embedded instances
      // namespace is omitted for embedded instances
      "classname": "CIM_Error",
      "properties": {
        "ErrorType": 4,
        "ErrorSource": "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\"",
        "ErrorSourceFormat": 2,
        "Message": "WIPG0213: CIM instance ACME_VirtualSystem.InstanceID=\"node47:s
ys11\" does not exist in CIM namespace root/cimv2.",
        "MessageArguments": [
            "ACME_VirtualSystem.InstanceID=\"node47:sys11\"",
            "root/cimv2",
            "GetInstance",
            null,
            "root/cimv2:ACME_VirtualSystem.InstanceID=\"node47:sys11\""
        ],
        "MessageID": "WIPG0213",
        "OwningEntity": "DMTF"
      }
    }
  ]
}
```

6.7 Representation of CIM-RS payload data types in JSON

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

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

6.8 Representation of CIM-typed values in JSON

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 `true` and `false`
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 *class* 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"},
```

```
  "AnOctetstringViaString": {
    "type": "string",
    "value": "0x00000007616263"},
  "AnOctetstringViaUint8Array": {
    "array": true,
    "type": "uint8",
    "value": [ 0, 0, 0, 7, 0x61, 0x62, 0x63 ]},
  "AnEmbeddedInstance": {
    "type": "instance",
    "value": . . . // Instance payload element, see 6.6.2
  },
  "AnEmbeddedObjectThatIsAnInstance": {
    "type": "instance",
    "value": . . . // Instance payload element, see 6.6.2
  },
  "AnEmbeddedObjectThatIsAClass": {
    "type": "class",
    "value": . . . // Class payload element, see 6.6.4
  },
  "ADatetime": {
    "type": "datetime",
    "value": "20120213175830.123456+060"},
  "AUint16": {
    "type": "uint16",
    "value": 20000},
  "ASint16": {
    "type": "sint16",
    "value": -16000},
  "AReal32": {
    "type": "real32",
    "value": 3.1415927},
  "ARef": {
    "type": "reference",
    "classname": "CIM_ComputerSystem",
    "value": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%
3Asys11"},
  "ABooleanArray": {
    "array": true,
    "type": "boolean",
    "value": [ true, false, null ]},
  "AFixedSizeBooleanArray": {
    "array": true,
    "arraysize": 3,
    "type": "boolean",
    "value": [ true, false, null ]},
  "AStringArray": {
    "array": true,
    "type": "string",
    "value": [ "some text", null, "more text\n" ]},
```

```
  "AReal64Array": {
    "array": true,
    "type": "real64",
    "value": [ 1E-42, "NaN", null, "-Infinity" ]},
  "ARefArray": {
    "array": true,
    "type": "reference",
    "classname": "CIM_ComputerSystem",
    "value": [
      "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%3Asys11
",
      null,
      "/root%2Fcimv2/classes/ACME_ComputerSystem/instances/InstanceID=node7"
    ]},
. . .
```

6.8.1 Representation of CIM real32 and real64 datatypes

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

- the IEEE 754 special values *positive infinity*, *negative infinity*, and any *not-a-number* values shall be represented as JSON String-typed values using the following strings:

  positive infinity: `"Infinity"`

  negative infinity: `"-Infinity"`

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

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

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

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

Examples:

```
. . .
  "Throughput": {
    "type": "real32",
    "value": 3.45E3},
  "ErrorRate": {
    "type": "real32",
    "value": "NaN"},
  "LowerLimit": {
```

```
    "type": "real64",
    "value": "-Infinity"},
. . .
```

6.8.2 Representation of CIM references

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

Example:

```
. . .
  "System": {
    "type": "reference",
    "classname": "CIM_ComputerSystem",
    "value": "/root%2Fcimv2/classes/ACME_VirtualSystem/instances/InstanceID=node47%
3Asys11"}
. . .
```

6.8.3 Representation of CIM Null values

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

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

Example:

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

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: • Added representation of data type • Upgraded to version 2.0 of generic operations (DSP0223) • Several other changes

Bibliography

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

DMTF DSP2032, *CIM-RS White Paper 2.0*,
http://www.dmtf.org/standards/published_documents/DSP2032_2.0.pdf

ECMA-262, *ECMAScript Language Specification, Edition 5.1*, June 2011,
http://www.ecma-international.org/publications/standards/Ecma-262.htm

ECMA-404, *The JSON Data Interchange Format, 1st Edition*, October 2013,
http://www.ecma-international.org/publications/standards/Ecma-404.htm

IANA MIME Media Types,
http://www.iana.org/assignments/media-types/

IETF RFC4627 (Informational), *The application/json Media Type for JavaScript Object Notation (JSON)*, July 2006,
http://tools.ietf.org/html/rfc4627

J. Holzer, *RESTful Web Services and JSON for WBEM Operations*, Master thesis, University of Applied Sciences, Konstanz, Germany, June 2009,
http://mond.htwg-konstanz.de/Abschlussarbeiten/Details.aspx?id=1120