
Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol — An Interface for Managing Cloud Infrastructure

*Model d'interface de management de l'infrastructure du nuage
informatique (CIMI) et protocole RESTful basé HTTP — Une interface
pour le management de l'infrastructure du nuage informatique*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2015

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

CONTENTS

34	Foreword	7
35	1 Scope	9
36	1.1 Document structure	9
37	1.2 Document versioning scheme	9
38	1.3 Typographical conventions	9
39	2 Normative references	10
40	3 Terms and definitions	11
41	4 HTTP-based protocol	14
42	4.1 Introduction	14
43	4.1.1 Protocol evolution and client expectations	14
44	4.1.2 XML namespaces	14
45	4.1.3 URI space	14
46	4.1.4 Media types	15
47	4.1.5 Request headers	15
48	4.1.6 Request query parameters	15
49	4.2 Protocol operations	21
50	4.2.1 Common CRUD operations	22
51	4.3 OVF support	29
52	5 Model	30
53	5.1 Resource wrappers	30
54	5.2 Extensibility	31
55	5.3 Identifiers	31
56	5.4 Attribute constraints	32
57	5.5 Data types and their serialization	33
58	5.5.1 boolean	33
59	5.5.2 dateTime	33
60	5.5.3 duration	33
61	5.5.4 integer	34
62	5.5.5 string	34
63	5.5.6 ref	34
64	5.5.7 map	35
65	5.5.8 structure	35
66	5.5.9 byte[]	36
67	5.5.10 URI	36
68	5.5.11 Arrays	36
69	5.5.12 Collections	37
70	5.5.13 "Any" type	41
71	5.5.14 Empty attribute values	41
72	5.6 Units	41
73	5.7 Relationship semantics	42
74	5.8 Operations	42
75	5.9 Alternative model formats	43
76	5.10 Resources	43
77	5.10.1 Common attributes	43
78	5.11 Resource metadata	45
79	5.11.1 Serialization of attribute value constraints	49
80	5.11.2 Capabilities	51
81	5.11.3 ResourceMetadataCollection Resource	54
82	5.12 Cloud Entry Point	55
83	5.12.1 Operations	61
84	5.13 System Resources and relationships	61
85	5.13.1 System	62

86	5.13.2	SystemCollection Resource	80
87	5.13.3	SystemTemplate Resource	81
88	5.13.4	SystemTemplateCollection Resource	87
89	5.14	Machine Resources and relationships	88
90	5.14.1	Machine	89
91	5.14.2	MachineCollection	107
92	5.14.3	MachineTemplate	109
93	5.14.4	MachineTemplateCollection Resource	116
94	5.14.5	MachineConfiguration Resource	117
95	5.14.6	MachineConfigurationCollection Resource	119
96	5.14.7	MachineImage Resource	120
97	5.14.8	MachineImageCollection Resource	124
98	5.14.9	Credential Resource	125
99	5.14.10	CredentialCollection Resource	126
100	5.14.11	CredentialTemplate Resource	127
101	5.14.12	CredentialTemplateCollection Resource	128
102	5.15	Volume Resources and relationships	130
103	5.15.1	Volume	131
104	5.15.2	VolumeCollection Resource	135
105	5.15.3	VolumeTemplate Resource	136
106	5.15.4	VolumeTemplateCollection Resource	138
107	5.15.5	VolumeConfiguration Resource	139
108	5.15.6	VolumeConfigurationCollection Resource	141
109	5.15.7	VolumeImage Resource	142
110	5.15.8	VolumeImageCollection Resource	144
111	5.16	Network Resources and relationships	145
112	5.16.1	Network	145
113	5.16.2	NetworkCollection Resource	153
114	5.16.3	NetworkTemplate Resource	154
115	5.16.4	NetworkTemplateCollection Resource	156
116	5.16.5	NetworkConfiguration Resource	157
117	5.16.6	NetworkConfigurationCollection Resource	158
118	5.16.7	NetworkPort	160
119	5.16.8	NetworkPortCollection Resource	164
120	5.16.9	NetworkPortTemplate Resource	165
121	5.16.10	NetworkPortTemplateCollection Resource	168
122	5.16.11	NetworkPortConfiguration Resource	169
123	5.16.12	NetworkPortConfigurationCollection Resource	170
124	5.16.13	Address Resource	171
125	5.16.14	AddressCollection Resource	173
126	5.16.15	AddressTemplate Resource	174
127	5.16.16	AddressTemplateCollection Resource	176
128	5.16.17	ForwardingGroup Resource	177
129	5.16.18	ForwardingGroupCollection Resource	180
130	5.16.19	ForwardingGroupTemplate Resource	181
131	5.16.20	ForwardingGroupTemplateCollection Resource	182
132	5.17	Monitoring Resources and relationships	183
133	5.17.1	Job Resource	184
134	5.17.2	JobCollection Resource	188
135	5.17.3	Meter Resource	189
136	5.17.4	MeterCollection Resource	195
137	5.17.5	MeterTemplate Resource	196
138	5.17.6	MeterTemplateCollection Resource	197
139	5.17.7	MeterConfiguration Resource	198
140	5.17.8	MeterConfigurationCollection Resource	201
141	5.17.9	EventLog Resource	202

142	5.17.10 EventLogCollection Resource	205
143	5.17.11 EventLogTemplate Resource	206
144	5.17.12 EventLogTemplateCollection Resource	207
145	5.17.13 Event Resource	208
146	6 Security considerations	216
147	ANNEX A (normative) OVF support in CIMI	217
148	ANNEX B (informative) XML Schema.....	219
149	ANNEX C (informative) Change log.....	220
150		

151 Figures

152	Figure 1 - Cloud Entry Point.....	56
153	Figure 2 - System Resources.....	62
154	Figure 3 - Machine Resources	89
155	Figure 4 - Volume Resources	130
156	Figure 5 - Network Resources	145
157	Figure 6 - Monitoring Resources.....	184
158		

159 Tables

160	Table 1 – XML namespaces	14
161	Table 2 – Named structure.....	35
162	Table 3 – Converting a relative URI to an absolute URI	36
163	Table 4 – Numerical equivalents for attributes.....	42
164	Table 5 – Common attributes.....	43
165	Table 7 – Capability URIs	51
166	Table 8 – CloudEntryPoint attributes	56
167	Table 9 – System attributes	63
168	Table 10 – SystemSystem attributes	67
169	Table 11 – SystemMachine attributes.....	68
170	Table 12 – SystemCredential attributes	70
171	Table 13 – SystemVolume attributes	71
172	Table 14 – SystemNetwork attributes	72
173	Table 15 – SystemNetworkPort attributes	74
174	Table 16 – SystemAddress attributes	75
175	Table 17 – SystemForwardingGroup attributes	76
176	Table 18 – SystemTemplate attributes	82
177	Table 19 – Machine attributes.....	89
178	Table 20 – Disk attributes	93
179	Table 21 – MachineVolume attributes	94
180	Table 22 – MachineNetworkInterface attributes	96
181	Table 23 – MachineNetworkInterfaceAddress attributes	98
182	Table 24 – MachineSnapshot attributes	100
183	Table 25 – MachineTemplate attributes.....	109
184	Table 26 – MachineConfiguration attributes	117
185	Table 27 – MachineImage attributes.....	120

ISO/IEC 19831:2015 (E)

186	Table 28 – Credential attributes	125
187	Table 29 – UserName/Password attributes	125
188	Table 30 – Public key attributes	125
189	Table 31 – CredentialTemplate attributes	127
190	Table 32 – Volume attributes	131
191	Table 33 – VolumeVolumeImage attributes	133
192	Table 34 – VolumeTemplate attributes	136
193	Table 35 – VolumeConfiguration attributes	140
194	Table 36 – VolumeImage attributes	142
195	Table 37 – Network attributes	145
196	Table 38 – NetworkTemplate attributes	154
197	Table 39 – NetworkConfiguration attributes	157
198	Table 40 – NetworkPort attributes	160
199	Table 41 – NetworkPortTemplate attributes	165
200	Table 42 – NetworkPortConfiguration attributes	169
201	Table 43 – Address attributes	171
202	Table 44 – AddressTemplate attributes	174
203	Table 45 – ForwardingGroup attributes	178
204	Table 46 – ForwardingGroupNetwork attributes	179
205	Table 47 – ForwardingGroupTemplate attributes	181
206	Table 48 – Job attributes	185
207	Table 49 – Meter attributes	189
208	Table 50 – Sample attributes	192
209	Table 51 – MeterTemplate attributes	196
210	Table 52 – MeterConfiguration attributes	198
211	Table 53 – aspect URIs	200
212	Table 54 – EventLog attributes	202
213	Table 55 – EventLogTemplate attributes	206
214	Table 56 – Event attributes	208
215	Table 57 – type URIs	211
216		
217		

Foreword

The *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol* specification (DSP0263) was prepared by the DMTF Cloud Management Working Group. It defines a logical model for the management of resources within the Infrastructure as a Service domain.

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

Acknowledgments

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

Editors (past and present):

- Marios Andreou – Red Hat
- Doug Davis – IBM
- Jacques Durand – Fujitsu
- Gilbert Pilz – Oracle

Contributors:

- Ghazanfar Ali – ZTE Corporation
- Marios Andreou – Red Hat
- Keith Bankston – Microsoft Corporation
- Winston Bumpus – VMware Inc.
- Nathan Burkhart – Microsoft Corporation
- Mark Carlson – Oracle
- Steve Carter – Novell
- Junsheng Chu – ZTE Corporation
- Josh Cohen – Microsoft Corporation
- Derek Coleman – Hewlett-Packard Company
- John Crandall – Brocade Communications Systems
- Doug Davis – IBM
- Jim Davis – WBEM Solutions
- Fernando de la Iglesia – Telefónica
- Hiroshi Dempo – NEC Corporation
- Jacques Durand – Fujitsu
- Yigal Edery – Microsoft Corporation
- George Ericson – EMC
- Colleen Evans – Microsoft Corporation
- Norbert Floeren – Ericsson AB
- Robert Freund – Hitachi, Ltd.
- Fermín Galán – Telefónica
- Krishnan Gopalan – Microsoft Corporation
- Kazunori Iwasa – Fujitsu
- Mark Johnson – IBM
- Bhumi Khasnabish – ZTE Corporation
- Dies Köper – Fujitsu
- Vincent Kowalski – BMC Software
- Ruby Krishnaswamy – France Telecom Group
- Lawrence Lamers – VMware Inc.
- Paul Lipton – CA Technologies
- James Livingston – NEC Corporation
- Vince Lubsey – Virtustream Inc.

ISO/IEC 19831:2015 (E)

265	• David Lutterkort – Red Hat
266	• Fred Maciel – Hitachi, Ltd.
267	• Andreas Maier – IBM
268	• Ashok Malhotra – Oracle
269	• Jeff Mischkinsky – Oracle
270	• Jesus Molina – Fujitsu
271	• Efraim Moscovich – CA Technologies
272	• Bryan Murray – Hewlett-Packard Company
273	• Steven Neely – Cisco
274	• Ryuichi Ogawa – NEC Corporation
275	• John Parchem – Microsoft Corporation
276	• Shishir Pardikar – Citrix Systems Inc.
277	• Miguel Peñalvo – Telefónica
278	• Gilbert Pilz – Oracle
279	• Alvaro Polo – Telefónica
280	• Enrico Ronco – Telecom Italia
281	• Federico Rossini – Telecom Italia
282	• Matthew Rutkowski – IBM
283	• Tom Rutt – Fujitsu
284	• Hemal Shah – Broadcom
285	• Nihar Shah – Microsoft Corporation
286	• Alan Sill – Texas Tech University
287	• Zhexuan Song – Huawei
288	• Marvin Waschke – CA Technologies
289	• Eric Wells – Hitachi, Ltd.
290	• Jeff Wheeler – Huawei
291	• Maarten Wiggers – Fujitsu
292	• Daniel Wilson – Ericsson AB
293	• Steve Winkler – SAP AG
294	• Jack Yu – Oracle
295	• Aaron Zhang – Huawei
296	• HengLiang Zhang – Huawei
297	

Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol

1 Scope

This specification describes the model and protocol for management interactions between a cloud Infrastructure as a Service (IaaS) Provider and the Consumers of an IaaS service. The basic resources of IaaS (machines, storage, and networks) are modeled with the goal of providing Consumer management access to an implementation of IaaS and facilitating portability between cloud implementations that support the specification. This document specifies a Representational State Transfer (REST)-style protocol using HTTP. However, the underlying model is not specific to HTTP, and it is possible to map it to other protocols as well.

CIMI addresses the management of the lifecycle of infrastructure provided by a Provider. CIMI does not extend beyond infrastructure management to the control of the applications and services that the Consumer chooses to run on the infrastructure provided as a service by the Provider. Although CIMI may be to some extent applicable to other cloud service models, such as Platform as a Service ("PaaS") or Storage as a Service ("SaaS"), these uses are outside the design goals of CIMI.

1.1 Document structure

This document defines a model and a RESTful HTTP-based protocol.

The core REST patterns are defined first and, after each resource is defined, any HTTP-specific information for that resource is specified.

1.2 Document versioning scheme

This document adheres to the versioning scheme defined in clause 6.3 of [DSP4004](#).

As the specification changes over time certain features might be deprecated. These are identified in the specification and should not be supported. Each of these deprecated features is clearly denoted in the clause in which they were previously defined.

1.3 Typographical conventions

This specification uses the following conventions:

In the narrative text of the specification:

- The regular or narrative font is Arial.
- Proper CIMI nouns such as Resource names, attribute names, operation names, reserved variable names are in Courier font. (e.g. Machine, volumes, \$expand). The plural form applies to such names to indicate several instances of such Resources (e.g. Machines, Systems).
- Examples text are in small Courier font and over a darker background.
- Quotes are used for any text that needs be distinguished as name or value of a particular concept (e.g. the "value constraints" attribute, the "Resource Name" column, a "false" value). In such cases, the string in quotes is always qualified by the concept it is an instance of.
- Names for CIMI concepts that may be common English words but have a very specific meaning in CIMI, are in narrative font but capitalized, e.g. Provider, Consumer, Resource, Collection.

ISO/IEC 19831:2015 (E)

When used in their common English sense they remain lower-case. However, CIMI modeling concepts that are used in a commonly understood manner remain in lower-case, such as: attribute, operation.

Inside tables describing the Resource data model:

- The narrative font is used for all terms, as the table structure qualifies them sufficiently.
- Where textual descriptions are introduced, the rules for narrative text apply.
- If a name is used as types (i.e., names of embedded structures as well as atomic types such as "integer", "string"), are in *italic*.
- Names that are just placeholders for actual names that may vary with each model instance, are between < > (e.g., <componentTemplate>).

Where the serialization of Resources is described, a pseudo-schema notation is used with the following conventions:

- Values in *italics* indicate data types instead of literal values.
- Characters are appended to items to indicate cardinality:
 - "?" (0 or 1)
 - "*" (0 or more)
 - "+" (1 or more)
- Vertical bars, "|", denote choice. For example, "a|b" means a choice between "a" and "b".
- Parentheses, "(" and ")", are used to indicate the scope of the operators "?", "*", "+" and "|".
- Ellipses (i.e., "...") indicate points of extensibility. Note that the lack of an ellipses does not mean no extensibility point exists, rather it is just not explicitly called out - usually for the sake of brevity.

Operation names Create, Update, Delete, Read are abstract operations that convey the semantics of concrete corresponding operations, such as HTTP methods or CIMI operation URIs.

2 Normative references

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

DMTF DSP0223, *Generic Operations 1.0*,
http://www.dmtf.org/standards/published_documents/DSP0223_1.0.pdf

DMTF DSP0243, *Open Virtualization Format Specification 1.1*,
http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.1*,
http://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf

DMTF DSP4004, *DMTF Release Process 2.4*,
http://www.dmtf.org/sites/default/files/standards/documents/DSP4004_2.4.pdf

IANA HTTP Header Registry, <http://www.iana.org/assignments/message-headers/perm-headers.html>

- 374 IEC 80000-13:2008, International Organization for Standardization, Geneva, Switzerland, *Quantities and*
 375 *units – Part 13: Information science and technology*, April 2008,
 376 http://www.iso.org/iso/catalogue_detail?csnumber=31898
- 377 IETF RFC2616, R. Fielding et al, *Hypertext Transfer Protocol -- HTTP/1.1*,
 378 <http://www.ietf.org/rfc/rfc2616.txt>
- 379 IETF RFC3986, T. Berners-Lee et al, *Uniform Resource Identifiers (URI): Generic Syntax*, August 1998,
 380 <http://www.ietf.org/rfc/rfc3986.txt>
- 381 IETF RFC4627, D. Crockford, *The application/json Media Type for JavaScript Object Notation (JSON)*,
 382 July 2006, <http://www.ietf.org/rfc/rfc4627.txt>
- 383 IETF RFC5246, T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*,
 384 <http://www.ietf.org/rfc/rfc5246.txt>
- 385 ISO 8601:2004, International Organization for Standardization, Geneva, Switzerland, *Data elements and*
 386 *interchange formats -- Information interchange -- Representation of dates and times*, March 2008,
 387 http://www.iso.org/iso/catalogue_detail.htm?csnumber=40874
- 388 ISO/IEC 14977:1996, Roger S. Scowen, *Extended BNF — A generic base standard*. Software
 389 Engineering Standards Symposium 1993.
 390 http://www.iso.org/iso/catalogue_detail?csnumber=26153
- 391 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
 392 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>
- 393 NIST Special Publication 800-145, Peter Mell and Timothy Grance, *The NIST Definition of Cloud*
 394 *Computing*, Sept. 2011, <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- 395 NIST Special Publication 500-292, Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee
 396 Badger and Dawn Leaf, *NIST Cloud Computing Reference Architecture*, Sept. 2011,
 397 [http://collaborate.nist.gov/twiki-cloud-](http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_SP_500-292_-_090611.pdf)
 398 [computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_SP_500-292_-_090611.pdf](http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_SP_500-292_-_090611.pdf)
- 399 Representational State Transfer, Roy Fielding, Doctoral dissertation, University of California, *Architectural*
 400 *Styles and the Design of Network-based Software Architectures (Chapter 5)*, 2000,
 401 http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- 402 XMLSchema - Part 1, World Wide Web Consortium (W3C) Recommendation, H. Thompson, et al.,
 403 Editors, *XML Schema Part 1: Structures Second Edition*, 28 October 2004,
 404 <http://www.w3.org/TR/xmlschema-1/>
- 405 XMLSchema - Part 2, World Wide Web Consortium (W3C) Recommendation, P. Biron, A. Malhotra,
 406 Editors, *XML Schema Part 2: Datatypes (Second Edition)*, 28 October 2004,
 407 <http://www.w3.org/TR/xmlschema-2/>

408 3 Terms and definitions

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

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

ISO/IEC 19831:2015 (E)

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 [DSP4004](#), [DSP0223](#), and [DSP1001](#) apply to this document. The following additional terms are used in this document.

3.1 authentication

The process of verifying a claim, made by a subject, that it should be allowed to act on behalf of a given principal (person, service, etc.). Typical authentication mechanisms involve the use of username/password combination or public/private key pairs.

3.2 authorization

The process of verifying that an authenticated principal (person, service, etc.) has permission to perform certain operations (e.g., read, update) on specific Resources. (Also known as Access Control.)

3.3 cloud

Synonymous with "cloud computing" as defined in section 2 of the NIST Definition of Cloud Computing [\[SP800-145\]](#).

3.4 Cloud Service Consumer

A category of actors that includes the Consumer Business Manager (who approves business and financial expenditures for consumed services; accounts for used service instances; establishes business relationships; sets up accounts, budget, and terms; etc.); the Consumer Service Administrator (who requests service instances and changes to service instances; purchases services within the business relationship; creates Service Users (including policies); allocates resources, such as computer and storage; generates reports, such as usage; etc.); and Service Users (who use service instances provided by a Cloud Service Provider). The term "Consumer" is used if the indicated action or activity could involve one or more of the above actors. In cases where the distinction between the actors in this category is relevant, the more detailed term is used.

For purposes of comparison and alignment, it should be noted that a Cloud Service Consumer is equivalent to the "Cloud Consumer" actor defined in the NIST Reference Architecture [\[SP500-292\]](#).

3.5 Cloud Service Provider

A category of actors that includes the Service Operations Manager (who manages the technical infrastructure required for providing cloud services; monitors and measures performance and utilization against SLAs; provides reports from monitoring and measurement; etc.); Service Business Manager (who offers all types of services developed by cloud service developers; accounts for services potentially offered by service Providers themselves and services offered on behalf of cloud service developers; establishes a portfolio of business relationships; and sets up accounts and terms for Consumers, etc.); and Service Transition Manager (who enables a customer to use the cloud service, including "onboarding", integration, and process adoption; defines and creates service offerings based on Templates and Configurations that can be used by Consumers and are populated into the catalog; etc.). The term "Provider" is used if the indicated action or activity could involve one or more of the above actors. In cases where the distinction between the actors in the category is relevant, the more detailed term is used.

For purposes of comparison and alignment, it should be noted that a Cloud Service Provider is equivalent to the “Cloud Provider” actor defined in the NIST Reference Architecture [\[SP500-292\]](#).

3.6 Collection

A particular kind of Resource that contains a collection of other Resources and has a representation and serialization defined in this specification. Synonym for “CIMI collection”.

3.7 Configuration

A set of metadata, the values of which serve as the parameters of a discrete conformation of a specific type of virtual resource.

3.8 Infrastructure as a Service (IaaS)

A cloud computing service model defined in section 2 of the NIST Definition of Cloud Computing [\[SP800-145\]](#).

3.9 message confidentiality

A quality of a message that prevents anyone but the intended receiver(s) from viewing its contents.

3.10 message integrity

A quality of a message that allows a receiver of that message to determine whether the contents of the message have been altered since its creation.

3.11 Resource

A representation of an entity managed by the [Cloud Service] Provider that is generally available to the [Cloud Service] Consumer to access or operate on by the way of the interface described in this specification. Synonym for “CIMI resource”.

3.12 Template

Synonym for “CIMI template”. A Resource that represents the set of metadata and instructions used to instantiate some other Resource (e.g., a MachineTemplate is used to create Machines). Templates may aggregate other metadata Resources such as other Templates, Configurations, and Images. For example, a MachineTemplate refers to a MachineConfiguration and a MachineImage.

How a specific protocol mapping, or implementation, chooses to supply Templates as inputs to the instantiation process may vary. However, some common patterns should be considered:

1. By reference - allow Consumers to reference a Template (that exists as a Resource in the Provider) as part of the instantiation operation.
2. By value - allow Consumers to dynamically provide the Template information as part of the instantiation operation.
3. Reference with overrides - allow Consumers to reference a Template (that exists as a Resource in the Provider) and provide additional values that override the attributes of that Template as part of the instantiation operation.

4 HTTP-based protocol

4.1 Introduction

All operations are based on the *HyperText Transfer Protocol (HTTP)*, version 1.1 [RFC2616]. Each request is sent by using an HTTP verb such as PUT, GET, DELETE, HEAD, or POST and includes a message body in either JSON or XML format. Each response uses a standard HTTP status code, whose semantics are interpreted in the context of the particular request that was made. Each Resource in the model has a MIME type that further contextualizes the payload of requests and responses.

Resources in the model are identified by URIs, and each Resource's representation shall contain an "ID" attribute, of type URI, that acts as a "self pointer." This URI shall be unique within the context of the Provider's implementation. Dereferencing (through an HTTP GET) the URI of a Resource yields a representation of the Resource containing attributes and links to associated Resources. To begin operations, a client shall know the URI to the main entry point of a Provider - also known as the "Cloud Entry Point" Resource. All other Resources within the environment shall then be discoverable by the way of the iterative following of links to associated Resources within each Resource retrieved.

4.1.1 Protocol evolution and client expectations

Future versions of this specification structure changes in such a way that clients that conform to an earlier version of this specification continue to work, and are not be adversely affected by the evolution of the protocol. Clients are expected to follow a few simple rules to ensure this compatibility:

1. Clients shall not assume that the serializations shown for responses in this specification are complete. In particular, clients shall accept responses that contain data mixed in with the serializations shown here, and shall ignore such data. However, per clause 4.2.1.3, clients shall include unknown data in PUT requests to update Resources.
2. Clients shall not assume anything about the operations supported by a server. They are expected to discover operations that are supported (and permissible) by navigating to Resources from the cloud entry point. The serializations of Resources encountered indicate which operations are supported by the server.

4.1.2 XML namespaces

Table 1 lists the XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 1 – XML namespaces

Prefix	XML Namespaces	Specification
cimi	http://schemas.dmtf.org/cimi/1	This specification
xs	http://www.w3.org/2001/XMLSchema	XML Schema Part2

4.1.3 URI space

While URIs returned by Providers are to be treated as opaque by Consumers, and Consumers shall not make assumptions about the layout of the URIs or the structures of the URIs for the Resources, a Consumer may augment URIs with any well-defined query parameters that are supported by the Provider as defined in clause 4.1.6.

The sample URIs used in this specification are not normative and the patterns used shall not be interpreted as guidance for implementations. For example, any of the following URIs might be used by Providers to reference a particular Machine Resource:

```

http://example.com/machines/12345
http://example.com/machines?id=12345
http://example.com/12345
http://example.com/Cloud/resource?id=12345

```

4.1.4 Media types

In this specification, Resource and response representations are encoded either in JSON, as specified in [RFC4627](#) or in XML. If serialized in JSON, the media-type for CIMI resources shall be "application/json". If serialized in XML, the media-type shall be "application/xml".

In the JSON serialization of CIMI representations sent by Providers, there shall be an additional attribute on the root object called "resourceURI" that contains the unique URI that is associated with the type of CIMI resource being serialized.

Note that this requirement applies even if the \$select attribute is used to subset the Resource being acted upon.

In the XML serialization of Collection representations sent by Providers there shall be a resourceURI attribute, as shown in the example XML serialization of Collections in clause 5.5.12.

This attribute is optional for Consumers to include. If included, this attribute's value shall match the "typeURI" attribute of the corresponding ResourceMetadata Resource (see clause 5.11), if ResourceMetadata is supported. This value shall also be equivalent to the wrapping element of the XML serialization; in other words, the namespace of the wrapper element concatenated a "/" and then its localName.

Any CIMI resource implemented by a Provider shall have representations in JSON and XML. The client implementation may thus use either JSON or XML in requests with any server implementation, and may request a specific serialization using server-driven content negotiation (using the Accept request header).

4.1.5 Request headers

This specification uses general-header, request-header, and entity-header headers as defined in [RFC2616](#) in request messages to provide metadata about the message. Applications using messages defined in this specification shall use headers consistent with the requirements of [RFC2616](#).

4.1.6 Request query parameters

Providers may choose to include query parameters as part of the URIs returned to Consumers. Consumers shall include those query parameters when sending messages to those URIs. If Providers choose to define query parameters care should be taken to avoid conflicts with CIMI defined query parameters.

To modify the behavior of the Provider when processing request messages, Consumers may augment request URIs as described in the following clauses. As stated in clause 4.1.3, URIs returned from Providers are to be treated as opaque by Consumers; however, it is the responsibility of the Consumer to understand the use of the query parameters defined in the following clauses and ensure correctness when making a request.

Unsupported, or unknown, query parameters shall be silently ignored by Providers. Consumers may examine the CloudEntryPoint's capabilities to determine whether support of these query parameters is enabled.

4.1.6.1 Filtering Collections

If retrieving the representation of a Collection, Consumers may include the `$filter` query parameter to reduce the number of entries of the Collection that are returned based on the data within the entries of the Collection. Providers shall interpret and process the `$filter` query parameter as described in this section. The `$filter` parameter shall be of the form:

```
?$filter=expression
```

where "expression" represents a mathematical expression denoting how the top-level attributes of the Resources within the Collection shall be filtered. The expression is defined by the following EBNF grammar:

```
Filter      ::= AndExpr ( 'or' Filter )* ;
AndExpr     ::= Comp ( 'and' AndExpr )*
Comp        ::= Attribute Op Value
              | Value Op Attribute
              | PropExpr
              | '(' Filter ')'
Op          ::= '<' | '<=' | '=' | '>=' | '>' | '!='
Attribute   ::= ? resource attribute name ?
Value       ::= IntValue | DateValue | StringValue | BoolValue
IntValue    ::= /[0-9]+/
DateValue   ::= ? as defined by XML Schema ?
StringValue ::= "..." | '...'
BoolValue   ::= 'true' | 'false'
PropExpr    ::= 'property[' StringValue ']' Op StringValue
```

Where PropExpr is used to find Resources that contain a property with a certain key/value combination. The key is the StringValue within the square brackets ([]) and the value is the StringValue after the Op. The Resource shall be considered to satisfy the search criteria if any of the properties in the Resources match the specified PropExpr.

Each of these shall be percent encoded in the URL as appropriate.

The choice of which operator (including 'and' and 'or') is limited based on the type of the value and attribute. The following example describes the allowable operators:

```
'or', 'and'           : Boolean value/attribute
'<', '<=', '=', '>=', '>', '!=' : Integer and date value/attribute
'=', '!='            : String value/attribute
```

Consumers may include multiple filters within a single URI. Provider shall treat multiple filters as a series of "and" expressions where an entry of the Collection shall only be included in the response message if it satisfies all of the filter expressions specified.

Examples:

In the following examples, the following sample base URIs are used.

The URI to the MachineCollection of the Cloud Entry Point is as follows:

```
/machines
```

623 The URI to a Machine is as follows:

624 `/machines/123`

625 The URI to the DiskCollection of a Machine is as follows:

626 `/machines/123/disks`

627 The URI to the MachineVolumeCollection of a Machine is as follows:

628 `/machines/123/volumes`

629 To filter the MachineCollection so that just Machines with a "name" attribute of "mine" are
630 returned, use the following filter:

631 `GET /machines?$filter=name='mine'`

632 To filter a DiskCollection of a Machine so that just Disks with a format of "ntfs" are returned, the
633 following filter would be used:

634 `GET /machines/123/disks?$filter=format='ntfs'`

635 If the \$filter attribute is used, the Collection's "count" attribute shall contain the number of Resources
636 matching the filter expression.

637 4.1.6.2 Subsetting Collections

638 If retrieving the representation of a Collection, Consumers may include query parameters to subset the
639 number of entities of the Collection that are returned. Providers shall interpret and process these query
640 parameters as described in this section. While the previous clause discussed how to perform a filter over
641 the data within the Collection, this clause uses ordinal position within the Collection to achieve the desired
642 reduction.

643 This specification defined two query parameters that, if used, shall indicate the first and last ordinal
644 positions of the entities within the Collection that are returned. The query parameters shall be of the form:

645 `?$first=number`

646 `?$last=number`

647 Where "\$first" indicates the (1-based) ordinal position of the first entity of the Collection to return and
648 "\$last" indicates the (1-based) ordinal position of the last entity of the Collection to return. Consumers
649 are not required to use both at the same time. If \$first is specified but \$last is not, the implied value
650 for \$last shall be the ordinal position of the last entity in the Collection. Conversely, if \$last is specified
651 but \$first is not, the implied value for \$first shall be 1.

652 If any part of the range as expressed by \$first and \$last is outside of the bounds of the Collection,
653 just the Resources (if any) in the Collection that are contained within that range shall be returned. A fault
654 shall not be generated if any part, or all, of the expressed range is outside the bounds of the Collection.
655 Note that if \$first is larger than \$last, the range shall represent an empty range and therefore no
656 Resources are returned.

657 If either \$first or \$last are specified, and a filter expression (as defined in clause 4.1.6.1) is also
658 specified, the filter expression shall be performed first and then the ordinal constraints of \$first and
659 \$last shall be applied.

660 4.1.6.3 Subsetting Resources

661 If retrieving the representation of a Resource, Consumers may include the \$select query parameter to
662 specify a subset of the Resource to be acted upon. Providers shall interpret and process this query
663 parameter as described in this section. This subsetting shall have the semantic equivalence of

referencing a different Resource whose attributes are a subset of the original Resource as specified by the attribute names listed in the \$select query parameter. The format of a \$select query parameter is:

```
?$select=attributeName,...
```

The value of the \$select query parameter shall be a comma separated list of top-level attribute names of the Resource, possibly including the string "operations" in case the intent is to select the operations available to the Consumer for this Resource. Any attribute name erroneously appearing in the list that is not part of the Resource shall be ignored by the Provider. An attribute name of "*" is equivalent to specifying all of the attributes of the Resource including its operations. Any attribute name explicitly appearing more than once in a URI shall have its second (and subsequent) appearances ignored.

The \$select query parameter may appear more than once in a URI. This is semantically equivalent to all of the attribute names appearing as values of a single \$select query parameter. For example:

```
?$select=name&$select=state
```

is equivalent to:

```
?$select=name,state
```

The order of attribute names in the \$select query parameter is not relevant for serialization purposes. The attributes are serialized per the serialization rules/order as specified by the Resource definition.

Note that per clause 4.1.4, if a Resource representation is sent by a Provider it shall always include the resourceURI attribute even if it is not specified in the \$select query parameter.

For example, to subset the list of Machine attributes being acted upon to just the "name" and "description", the following query parameter would be used:

```
?$select=name,description
```

See clause 4.2.1.3.1 for more information about the impact of using this query parameter when updating a Resource.

If \$select is used in the URI for a Collection resource, the subsettings shall apply to the attributes of the Collection resource itself as for any other Resource. For example, to subset a Collection resource in order to only return the number of its items, plus the operations available on this Collection:

```
?$select=count,operations
```

However, exceptionally for Collection resources, if some attribute provided in the \$select list is not a top-level attribute of the Collection resource but instead is an attribute of the entities that are items of the Collection, the subsetting shall apply to each item of the Collection regarding this attribute. For example, if retrieving the DiskCollection, the following query parameter:

```
?$select=name,capacity
```

returns a collection of the Disks associated with a Machine but each entity of the collection just has the name and capacity attributes and nothing else, not even the operations or id attributes.

Optionally, an implementation may also support the alternative attribute name notation: <collectionName>/<attributeName> for subsetting the items inside a collection. For example, the following subsetting on items of a Disks Collection is equivalent to the one done in the previous example, while in addition listing the operations of the Collection resource itself (not of its items):

```
?$select=disks/name,disks/capacity,operations
```

This notation, if supported (see the "QueryPathNotation" capability in 5.11.2), allows for disambiguating subsettings if the same attribute name can be found for the Collection and for each item in the collection (which is always the case for id and operations).

4.1.6.4 Expanding references

If retrieving the representation of a Resource, Consumers may include the \$expand query parameter to specify which of the top-level "reference" attributes of the Resource shall be "expanded". Providers shall interpret and process this query parameter as described in this section. To expand a reference means that the attributes of the Resource being referenced shall be included in the serialization of that attribute. This feature allows for a more optimized retrieval of Resources.

The serialization shall be performed as follows:

JSON serialization:

```
"name": { "href": string }
```

shall be expanded to be:

```
"name": {
  "href": string,
  ... attributes of referenced resource...
}
```

XML serialization:

```
<name href="xs:anyURI"/>
```

shall be expanded to be:

```
<name href="xs:anyURI">
  ... attributes of the referenced resource...
</name>
```

Note that in the XML case the nested elements shall not contain the wrapper element of the referenced Resource (e.g., <Machine> in the case of a reference to a Machine Resource).

The format of a \$expand query parameter shall be:

```
?$expand=attributeName,...
```

The value of the \$expand query parameter is a comma-separated list of attribute names. Any attribute name erroneously appearing in the list that is not part of the Resource, or is not a reference, shall be ignored by the Provider. An attribute name of "*", or no attribute name list at all, is equivalent to specifying all of the attributes. Any attribute name explicitly appearing more than once in a URI shall have its second (and subsequent) appearances ignored.

The \$expand query parameter may appear more than once in a URI, which is semantically equivalent to all of the attribute names appearing as values of a single \$expand query parameter.

If the Resource being retrieved is a Collection, the attribute names listed in the \$expand shall apply to the attributes of the entities within the Collection. For example, specifying:

```
?$expand=volumes
```

if retrieving the MachineCollection has the same net effect as applying the "expand" semantics to the specified attribute ("volumes" in this example) of each Machine within the Collection. To be clear, \$expand acts on the attributes of the Resources in the Collection, not on the wrapping Collection Resource itself.

4.1.6.5 Specifying the Resource format

If retrieving the representation of a Resource, the HTTP Accept header is used to specify the encoding style of the response. While it is recommended that Consumers use the Accept header, there might be situations where Consumers are unable to control the values specified in that header. In these cases Consumers may use the `$format` query parameter to override the Accept header values. Providers shall interpret and process the `$format` query parameter as described in this section.

The `$format` parameter shall be of the form:

```
?$format=encoding
```

Where "encoding" is the requested representation of the response. This specification defines two possible values: "json" and "xml". Provider may support others. The value of the `$format` query parameter shall be case insensitive.

If both an Accept header and `$format` query parameter are present in a request message, the `$format` value shall take precedence. If the `$format` query parameter appears more than once, the second, and subsequent, appearances shall be ignored.

4.1.6.6 Sorting Collections

If retrieving the representation of a Collection, Consumers may include the `$orderby` query parameter to sort the entries of the Collection that are returned based on different attributes or in a different order (descending). Providers shall interpret and process the `$orderby` query parameter as described in this section. The `$orderby` parameter shall be of the form:

```
?$orderby=attributeName[:asc|:desc], ...
```

The `$orderby` expression may include multiple, comma-separated attribute names. Each attribute name may be optionally followed immediately by a colon and "asc" to denote ascending order (default), or "desc" to denote descending order for that attribute. If neither asc nor desc is specified, the order shall be "ascending".

The attributes included in the `$orderby` shall be of the following types as defined in clause 5.5: boolean, dateFormat, duration, integer, or string.

The sort shall be performed based on the attribute type.

The following rules apply to the ascending sort order:

- boolean – 'false' shall come before 'true'.
- dateTime – Earlier datetime shall come before a later datetime.
- duration – A shorter duration shall come before a longer duration.
- integer – Smaller integer shall come before larger integers. Negative integers shall come before positive integers.
- string – Ordering is based on Unicode/UTF-8 sort order.

For the desc sort order, the reverse of the above shall be performed.

Examples:

To sort the result set of the `MachinesCollection` Resource on the "created" attribute in descending order, the following expression would be used:

```
GET /machines?$orderby=created:desc
```

To sort the result set of the `MachinesCollection` Resource on the "cpu" attribute in descending order, followed by the "memory" attribute in ascending order, the following expression would be used:

```
GET /machines?$orderby=cpu:desc,memory:asc
```

4.1.6.7 Response headers

As defined in [RFC2616](#), this specification uses general-header, response-header, and entity-header headers in response messages to provide metadata about the message. Applications that use messages defined in this specification shall use headers consistent with the IANA HTTP Header Registry.

4.1.6.8 Job header

If the server supports the Job Resource, response messages shall include a header defined by this specification to indicate the URI for the job created to process the associated request message.

```
CIMI-Job-URI = "CIMI-Job-URI" ":" string
```

In cases where an error occurs during the processing of a request, the Provider shall include a representation of a Job Resource describing the status of the failed operation. This representation of a Job shall be included even in cases where the Provider does not normally support Job Resources to ensure that Consumers are provided with sufficient information, in a consistent manner, as to the reason for the failure regardless of whether the Provider supports Jobs. If Jobs are not supported in general, any of the references in the Job representation (e.g., "id" or the "href" for `nestedJobs`) shall be empty paths (i.e., "") and the `nestedJobs` array shall be expanded (see 4.1.6.4) to inline the representation of the pseudo subordinate Jobs.

4.1.6.9 ETag support

An ETag header may be provided by a Provider with each Resource as specified in [RFC2616](#). If a Provider does provide an ETag header, it shall also support If-Match header processing on behalf of the Consumer.

4.2 Protocol operations

This clause defines the set of common HTTP operations that a Provider may expose. At its core, there are four basic CRUD (Create, Read, Update, and Delete) operations. The manner in which these are used is consistent across all Resources within the model; therefore, their use is defined once and is to be applied consistently. Some Resources support specialized operations that do not fit well into a CRUD style of operation and those follow a similar high-level pattern, but each operation is allowed to have slight variations to accommodate its specific needs. The specifics of these special operations are detailed within the clause that defines the Resource.

If appropriate, some of the Resource representations include an "operations" attribute. Providers shall only include the "operations" attribute if the specified operations are accessible to the current client for that particular Resource. This situation means that based on many factors (e.g., authorization rights of the clients, current state of the Resource, etc.), a different set of "operations" shall be returned on each serialization of the Resource. Each operation shall include a "rel" and an "href" field. The "rel" field shall uniquely identify the operation name (e.g., "add", "edit"), while the "href" field is the URI to which the operation's request message shall be sent. Note that the "href" field's URI may be different from the URI of the Resource itself. The operations attribute shall be serialized as follows:

JSON serialization:

```
{ "operations": [
  { "rel": "string", "href": "string" }, +
]
```

XML serialization:

```
<Resource xmlns="http://schemas.dmtf.org/cimi/1">
  <operation rel="xs:anyURI" href="xs:anyURI"/> *
</Resource>
```

For example, the "edit" operation would appear as:

JSON serialization:

```
{ "operations": [
  { "rel": "edit", "href": "<editURI>" }
]
```

XML serialization:

```
<Resource xmlns="http://schemas.dmtf.org/cimi/1">
  <operation rel="edit" href="<editURI>I"/>
</Resource>
```

Additional "rel" values may be defined by Providers; however, they shall be fully qualified URIs and not relative URIs.

4.2.1 Common CRUD operations

Each of the Resources supported by this protocol shall adhere to the interaction patterns defined in the following clauses.

4.2.1.1 Creating a new Resource

To create a new instance of a Resource type, an HTTP POST request is sent to a designated "addURI" for that Resource type. In many cases, the Collection resource that maintains, or groups, all instances of that Resource type includes an "add" operation. The "add" operation references the addURI that is to be used.

The HTTP POST request shall include:

- CIMI serialization of the request to create a new Resource in the HTTP Body
- HTTP Content-Type header
- HTTP Content-Length header

For example, the request can be:

```
POST <addURI> HTTP/1.1
Host: <hostname>
```

```

862 Accept: application/(json+xml)
863 Content-Type: application/(json+xml)
864 Content-Length: <length>
865
866 <serialization of request to create a new resource>

```

This example has an Accept header with one of the CIMI supported media types: application/json or application/xml. If the Provider chooses to reply with a serialization, this serialization should be of the specified media type. Omission of the Accept header allows the Provider to reply with a serialization of any media type. If the Resource has a "State" attribute, its value shall be "CREATING" while the Provider is processing this operation.

Many of the create requests are defined such that a Template of the new Resource is passed. These create requests allow for the Template to be passed in "by-reference" or "by-value." For example, creating a new Machine looks like this (here using XML):

```

875 <MachineCreate xmlns="http://schemas.dmtf.org/cimi/1">
876   <name> xs:string </name> ?
877   <description> xs:string </description> ?
878   <property key="xs:string"> xs:string </property> *
879   <machineTemplate href="xs:anyURI"? >
880     ... template attributes ... ?
881   </machineTemplate>
882 </MachineCreate>

```

Note that in the XML case the creation of a new Machine requires a wrapper element named MachineCreate per the rules specified in clause 5.5.12.1.

More generally, creating a new Resource shall follow one of these two serialization patterns (here illustrated in JSON):

(1) Resource creation by passing a Template by value:

```

888
889 { "resourceURI": "http://schemas.dmtf.org/cimi/1/ResourceCreate",
890   "name": "myResourceName", ?
891   "description": "My resource description", ?
892   "properties": { "prop1name" : "prop1value" , + }, ?
893   "resourceTemplate": {
894     <here the template is passed by value>
895   }
896 }

```

(2) Resource creation by passing a template by reference:

```

898
899 { "resourceURI": "http://schemas.dmtf.org/cimi/1/ResourceCreate ",
900   "name": "myResourceName", ?
901   "description": "My resource description", ?
902   "properties": { "prop1name" : "prop1value" , + }, ?
903   "resourceTemplate": { "href": string ,
904     <here some template attribute/value pairs may be added to override values
905 in the referenced template>
906   }

```

}

In case the created Resource is itself a Template, only the first creation pattern - by value - applies.

In both patterns (1) and (2) the resourceURI attribute specifies the operation here generically identified as "ResourceCreate", e.g., MachineCreate.

In both patterns (1) and (2) an element corresponding to the Resource Template (here identified generically as "resourceTemplate" e.g., MachineTemplate) is specifying the Template to be used, either by value (1) or by reference (2).

Direct setting of attributes in the new Resource:

In a creation request it is possible to set the value of some attributes of the newly created Resource, regardless of what values the Template instantiation might have set if used alone. Three common attributes of the new created Resource may be set: name, description, and properties.

The semantics shall be same as of a partial update of the Resource for these attributes (described in a next subsection), immediately following the Resource creation from the Template alone.

Defining or referring to the Resource Template:

In pattern (1) above, the Provider may choose to create a Template Resource from the value given, but such creation is temporal in nature. The Provider shall not expose such a transient Resource to the Consumer and no such transient Resource shall be included in any query results back to the Consumer.

In pattern (2) above, additional attribute name/value pairs may be given inside the ResourceTemplate element that also contains the reference to the external (pre-existing) Template in order to override similar attributes defined in the Template. More precisely:

- Any top-level attribute of complex or simple type in the referred Template shall be overridden by providing its name/value pair in the create request inside the resourceTemplate element and immediately under it. For a top-level attribute of complex type (e.g., arrays, Collections, structures), the provided complex value shall also set all underlying attributes – e.g., array elements.
- The semantics shall be same as of modifying (overriding) parts of the referred Template just before it is used for instantiation, but these overrides shall not persist in the referred Template and shall only concern this particular instantiation.

In pattern (2) above, Consumers may erase any Template attributes by specifying either

```
"attribute": null
```

for the attribute in the JSON serialization, or

```
<attribute/>
```

in the XML serialization for that attribute.

Examples:

Here is an example of creation pattern (1) using a MachineTemplate by value (in JSON):

```
{ "resourceURI": "http://schemas.dmtf.org/cimi/1/MachineCreate ",
  "name": "myMachine123",
  "description": "A machine to be connected to a pre-existing network",
  "machineTemplate": {
    <here a template passed "by value" i.e. the attribute/value pairs for the
    MachineTemplate template. An example is of the networkInterfaces below: >
```

```

949     "networkInterfaces": [
950       { "addresses": [ { "address": { "href": "http://example.com/addresses/add1"
951     }}, { "address": { "href": "http://example.com/addresses/add2" } } ],
952         "network": { "href": "http://example.com/networks/net1" },
953         "state": "ACTIVE" }
954     ]
955   }
956 }
957 }

```

In the previous example:

The attributes `name` and `description` are instance-level settings because they are outside the `machineTemplate` element (i.e., they set attribute values in the new created Resource, not in the Template used to create the Resource). The name of the new Machine is "myMachine123".

This Machine is connected to an existing Network of reference (`http://example.com/networks/net1`), as specified in the Template complex attribute.

Here is an example of creation pattern (2) using a MachineTemplate by reference:

```

965 { "resourceURI": "http://schemas.dmtf.org/cimi/1/MachineCreate ",
966   "name": "myMachine456",
967   "description": "A machine connected to a pre-existing volume",
968   "machineTemplate": { "href": "http://example.com/machineTemplates/72000",
969     "credential": { "href": "http://example.com/myCredential" }
970     "networkInterfaces": [
971       { "addresses": [ { "address": { "href": "http://example.com/addresses/add4"
972     }}, { "address": { "href": "http://example.com/addresses/add5" } } ],
973         "network": { "href": "http://example.com/networks/net1" },
974         "state": "ACTIVE" }
975     ]
976   }
977 }
978 }
979 }

```

In the above example, a new machine named "myMachine456" is created, also connected to the same existing Network as in example (1), but with a different set of Addresses. Two kinds of attributes are provided with values at creation time in this example:

- Instance-level attribute settings: these shall directly update similar attributes in the created Resource, here name and description.
- Template-level overrides: The referred MachineTemplate is used for creating the Machine, but the `credential` attribute in this Template is (temporarily) overridden by the credential provided in the creation request as is the `networkInterfaces` array. In case such attributes were not present in the referred Template, they are added (temporarily) just for this Machine creation.

Some of the create requests allow for configuration type of Resources to be passed by-reference or by-value as well - e.g., `Credential` on a Machine create operation. The processing rules defined above applies in those cases as well.

If the response has a 201 status code, the response shall include:

- HTTP Location header with a reference to the new Resource

ISO/IEC 19831:2015 (E)

995 If the response to a create request includes a serialization of the new Resource, the response shall
996 additionally include:

- 997 • HTTP Content-Type header
- 998 • HTTP Content-Length header

999 For example, the response can be:

```
1000 HTTP/1.1 201 Created
1001 Location: <location>
1002 Content-Type: application/(json|xml)
1003 Content-Length: <length>
1004
1005 <serialization of new resource>
```

1006 4.2.1.2 Retrieving a representation of a Resource

1007 To retrieve a representation of Resource, an HTTP GET request is sent to the Resource's URI.

1008 For example, the request can be:

```
1009 GET <ResourceURI> HTTP/1.1
1010 Host: <hostname>
1011 Accept: application/(json|xml)
```

1012 If the response has a 200 status code, the response shall include:

- 1013 • HTTP Content-Type header
- 1014 • HTTP Content-Length header

1015 For example, the response can be:

```
1016 HTTP/1.1 200 OK
1017 Content-Type: application/(json|xml)
1018 Content-Length: <length>
1019
1020 <serialization of resource>
```

1021 4.2.1.3 Updating a Resource

1022 To update a Resource's state, an HTTP PUT request containing the complete, updated representation is
1023 sent to a designated editURI for that Resource type. Consumers shall include all non-empty attributes
1024 of the Resource in the PUT request - including ones that it might not support or understand that were
1025 returned in a GET response. This is to ensure that a client does not inadvertently modify (erase) data in a
1026 Resource by excluding it from the full representation of the Resource.

1027 In many cases, this editURI is the same as the URI of Resource itself. Retrieving the Resource
1028 representation shall include an "edit" operation, which contains the editURI that is to be used, if the
1029 requester is allowed to modify the Resource.

1030 While processing a PUT request, if the server detects that an attempt is being made to update a
1031 read-only, or immutable, attribute, it shall silently ignore that attribute update request and shall not
1032 generate an error. This rule applies to Resource partial updates as well.

Because of potential conflicts that might occur due to multiple concurrent updates, Consumers should use the partial update mechanism, defined in 4.2.1.3.1, to reduce the chances of mistakenly updating attributes with out-of-date data.

The HTTP PUT request shall include:

- CIMI serialization of the updated Resource in the HTTP Body
- HTTP Content-Type header
- HTTP Content-Length header

For example, the request can be:

```
PUT <editURI> HTTP/1.1
Host: <hostname>
Accept: application/(json|xml)
Content-Type: application/(json|xml)
Content-Length: <length>

<serialization of request to update a resource>
```

If the response includes a serialization of the updated Resource and has a status code of 200, this response shall include:

- HTTP Content-Type header
- HTTP Content-Length header

For example, the response can be:

```
HTTP/1.1 200 OK
Content-Type: application/(json|xml)
Content-Length: <length>

<serialization of updated resource>
```

4.2.1.3.1 Partial updates to a Resource

For clarity, this clause explains how to use the \$select query parameter (see clause 4.1.6.3) to subset a Resource for the purposes of only operating on a selected set of top-level attributes.

To update only certain top-level attributes of a Resource, a Consumer may include only the altered attributes in the representation of the Resource within the HTTP request body. If this request is made, the URI to the Resource shall include the attributes to be modified as a comma-separated list of query parameters; in other words, the URI shall be of the form:

```
http://example.com/resource?$select=attribute1,attribute2,...
```

Only the attributes listed in the URI's query parameters shall be modified; attributes not listed in the URI shall not be directly modified by the request. Note that this circumstance does not preclude the modification of one attribute causing side-effects that result in the modification of an attribute not listed in the query parameters.

Any attribute listed in the URI but not included within the HTTP request body shall be reset to a Resource specific value (e.g., removed).

From an HTTP perspective, the updated subsetting Resource is a distinct one. The semantics of a normal HTTP PUT are adhered to; it is a complete replacement update of the specified Resource. From the Consumer's perspective, the partial update is interpreted and executed by the Cloud Service Provider, and some part of the Resource is changed.

Adhering to the generic PUT semantics defined previously, any attribute of the original (full) Resource included within the HTTP request body shall result in an error being generated if that attribute is not listed in the \$select query parameter - see clause 5.4. Note that this is due to these attributes being unknown to this subsetting Resource.

The following sample request updates just the name and description attributes of a Machine:

```
PUT /machines/myMachine?$select=name,description HTTP/1.1
Host: <hostname>
Accept: application/xml
Content-Type: application/xml
Content-Length: <length>

<Machine>
  <name>My New Machine</name>
</Machine>
```

The name attribute is set to "My New Machine" and the description attribute is erased.

4.2.1.4 Deleting a Resource

To delete a Resource, an HTTP DELETE request is sent to a designated deleteURI for that Resource type. In many cases, this deleteURI is the same as the URI of Resource itself. Retrieving the Resource representation shall include a "delete" operation, which contains the deleteURI that is to be used, if the requester is allowed to delete the Resource.

For example, the request can be:

```
DELETE <deleteURI> HTTP/1.1
Host: <hostname>
```

If the Resource has a State attribute, its value shall be "DELETING", while the Provider is processing this operation.

For example, the response can be:

```
HTTP/1.1 200 OK
```

4.2.1.5 Other operations

While some modifications to the Resources in the model can be done by the way of a simple update (PUT) operation to the Resource's editURI, sometimes a more complex set of actions needs to be taken. In these cases, the operations shall be modeled as HTTP POSTs to the operation specific URI of the Resource.

For each of the Resources that define additional operations, a description of the HTTP request and response bodies is provided. However, the general HTTP interaction are as described below.

The request shall be of the following form:

```
POST <operationLinkURI> HTTP/1.1
```

```

Host: <hostname>
Accept: application/(json+xml)
Content-Type: application/(json+xml)
Content-Length: <length>

<serialization of request to perform some action>

```

The form of the response varies depending on the operation and is defined by the operation itself.

Note that the definition of the Create operation (see clause 4.2.1.1) follows this same pattern. It is just called out for ease of reference.

4.2.1.6 Synchronous operations

If a Provider supports the Job Resource, each incoming PUT, DELETE, POST request shall result in a Job Resource being created and an absolute URI reference to that Job Resource shall be returned back to the client by the way of the CIMI-Job-URI HTTP Header in the HTTP response message:

```
CIMI-Job-URI: <uri-to-Job>
```

In this case, the requested operation shall be complete and the Job URI shall point to a completed Job. If the Job is not complete, the server shall return a 202 and follow the instructions for Asynchronous operations.

4.2.1.7 Asynchronous operations

In some cases, an operation requested by the client may take an undetermined amount of time to be completed. For example, creating a new Machine or starting an existing Machine may take a relatively long time to be completed. In these cases, it is not practical to complete these operations within a reasonable HTTP request timeout interval, so the Provider shall return an HTTP "202 Accepted" response code.

As with synchronous operations, if a Provider supports the Job Resource, it shall create a Job Resource for the incoming request and return a reference to that Job Resource back to the client by the way of the CIMI-Job-URI HTTP Header in the HTTP response message. Additionally, in the case of a "202 Accepted" response code, the Provider may also return any of the following in the HTTP response body:

- A representation of the Job Resource, if one was created.
- A partial representation of the response message as if the operation were a synchronous operation. For example, when creating a new Machine, the response message may include a partial representation of the new Machine in the response message. The list of attributes of the Resource that is returned is implementation specific and based upon how much information is available at the time the response message is generated, but it shall be consistent with the definition of the full Resource representation. In the case of a create operation, the Provider may also include an HTTP Location header referencing the "to be created" Resource, if it is known.
- An empty response body.

Note that the decision as to whether any particular operation is synchronous or asynchronous is at the server's discretion.

4.3 OVF support

The *Open Virtualization Format (OVF) Specification* ([DSP0243](#)) describes an open, secure, portable, efficient, and extensible format for the packaging and distribution of software to be run in virtual

machines. OVF support in CIMI allows an OVF package to be used to create CIMI management resources by importing the package. Additionally, CIMI management resources can be exported into an OVF package. The actual support for the OVF package is typically provided by a hypervisor that is managed by the CIMI provider. The import of an OVF package exposes CIMI specific constructs and parameters as a result of the import without altering the original OVF package. Thus the CIMI resources that are created as a result of the import form a “View” of what the hypervisor did; however, other (non-CIMI mapped) information from the OVF package may have been used by the hypervisor in its import. This other information is implementation dependent and is not further touched upon by this standard.

An OVF package can support single virtual machines (VMs) corresponding to a single CIMI Machine or MachineTemplate (see clause 5.14.1) or may also support a complex hierarchy of VMs and their related Resources corresponding to a CIMI System or SystemTemplate (see clause 5.13.1) and related CIMI management resources.

OVF support is covered in more detail in ANNEX A.

5 Model

This model assumes that a business relationship has already been established between the Consumer and the Provider. This relationship may include financial terms, creating separately administered clouds that the consuming organization is paying for, and the establishment of authentication credentials to access the administrative entry point for each cloud. The scope of this model is one separately administered cloud.

The CIMI model is described here by using a tabular representation. It is inspired from Entity-Relationship modeling, where each entity is modeling a significant cloud resource for which independent access and manipulation is expected. Relationships between resources use a referential mechanism based on unique identifiers that is expected to be already supported by the implementation environment and protocol (e.g., URIs for HTTP).

The model is self-describing and allows for querying its own metadata, e.g., to discover which extensions have been implemented. The model is also extensible in different ways (see clause 5.1).

Along with this model, a serialization of its entities is defined (both in XML and JSON).

An alternative UML diagram representation is provided for each major group of resources.

5.1 Resource wrappers

The serialization of Resource instances in the model follow these conventions. Consider the serialization of a Resource named “MyResource”:

JSON serialization:

The Resource is serialized as an object wrapping all its attributes, but without a wrapper name. The Resource includes a resourceURI with a URI for the type of Resource being serialized. For example:

```
{ "resourceURI": "http://example.com/MyResource",
  "attribute": "value"
}
```

XML serialization:

The Resource is serialized as an element with name equal to the Resource name; for example:

```
<MyResource xmlns="http://example.com">
  <attribute> value </attribute>
```

</MyResource>

5.2 Extensibility

There are two types of extensibility mechanisms defined by the CIMI model; one is intended for use by Consumers whilst the other is to be used by Providers.

The first allows for a CIMI Consumer to add additional data to a Resource. Each Resource in the CIMI model has an attribute called "properties". Consumers, when creating or updating a Resource, may store any name/value pair in the properties attribute. CIMI Providers shall store and return these values to the Consumer. There is no obligation for the Provider to understand or take any action based on these values; they are there for the Consumer's convenience. Providers shall not add elements to this properties attribute.

The second type of extensibility mechanism allows for Provider defined extensions and this specification includes the ResourceMetadata Resource for this purpose. ResourceMetadata may be used to

- express constraints on the existing CIMI defined Resource attributes (e.g., express a maximum for the 'cpu' attribute of the MachineConfiguration Resource)
- introduce new attributes for CIMI defined Resources together with any constraints governing these (e.g., a new 'location' attribute for the Volume Resource that takes values from a defined set of strings)
- introduce new operations for any of the CIMI defined Resources (e.g., define a new 'compress' operation for the Volume Resource)
- express any Provider specific capabilities or features (e.g., the length of time that a Job Resource is retained after Job completion and before this is deleted)

It is recommended that Providers use the ResourceMetadata Resource to advertise these attributes, operations, and capabilities along with any constraints that might need to be understood by Consumers. The ResourceMetadata Resource is defined in clause 5.11.

If a Provider receives a message containing an unknown or unsupported attribute, it shall reject the request. If a Consumer receives a message containing an unknown or unsupported attribute, it shall silently ignore the attribute. However, Consumers are required to include those attributes in messages sent back to the Provider. Note in these cases the Consumer is not required to understand or process the unsupported attribute, but merely echo it back to the Provider.

5.3 Identifiers

All identifiers (e.g., Resource names, attributes, operations, parameter names) defined by this specification, or defined by the way of an extension, shall adhere to the following rules:

- Identifier names shall be treated as case sensitive.
- Identifier names shall only use the following set of characters:
 - o Uppercase ASCII (U+0041 through U+005A)
 - o Lowercase ASCII (U+0061 through U+007A)
 - o Digits (U+0030 through U+0039)
 - o Underscore (U+005F)
- Identifier names shall not start with a Digit (U+0030 through U+0039).

1233 Note that these rules do not apply to the "name" common attribute defined in clause 5.10.1.

1234 **5.4 Attribute constraints**

1235 Each attribute of the Resources in the CIMI model is augmented by a set of constraints that further qualify
1236 the attribute that is being defined. For each attribute, there is a Provider and a Consumer set of
1237 constraints because each might differ. The following constraints are possible:

1238 **support optional:**

1239 This constraint indicates that support for this attribute is optional. If supported, Providers should advertise
1240 its support through ResourceMetadata. See clause 5.2 for information concerning the processing of
1241 unsupported and unknown attributes. See clause 5.5.14 regarding empty attribute values.

1242 Non-empty Consumer supported writeable (i.e., read-write and write-only) attributes shall always be
1243 included as part of the Resource representation sent from Consumers to Providers, including create
1244 requests.

1245 Non-empty Provider-supported attributes shall always be included as part of the Resource representation
1246 sent from Providers to Consumers.

1247 **support mandatory:**

1248 This constraint indicates that support for this attribute is required by compliant implementations. If present
1249 on a nested attribute, this attribute is required to be supported only if the parent attribute is supported.
1250 See clause 5.5.14 regarding empty attribute values.

1251 Non-empty mandatory writeable (i.e., read-write and write-only) attributes shall always be included as part
1252 of the Resource representation sent from Consumers to Providers - including create requests.

1253 Non-empty Provider mandatory attributes shall always be included as part of the Resource representation
1254 sent from Providers to Consumers.

1255 **immutable:**

1256 This Provider constraint indicates that the attribute, once set, shall never change for the lifetime of the
1257 Resource.

1258 **mutable:**

1259 This Provider constraint indicates that the attribute may be modified. Providers shall always have the
1260 ability to modify these attributes. Whether Consumers have the ability to modify these attributes shall be
1261 indicated by the read-only, read-write, and write-only constraints.

1262 **read-only:**

1263 This Consumer constraint indicates that the attribute may be retrieved but not updated by Consumers.
1264 Read-only attributes are not required to appear in the serialization of Resources in create or update
1265 request messages. If present, they shall be silently ignored by the Provider. Read-only attributes shall
1266 appear in the serialization of Resources sent from Providers.

1267 **read-write:**

1268 This Consumer constraint indicates that the attribute may be retrieved and/or updated by Consumers.
1269 Read-write attributes shall appear in the serialization of Resources sent to and from Providers. Providers
1270 may further constrain whether Consumers can update these attributes and should indicate this by the way
1271 of ResourceMetadata.

write-only:

This Consumer constraint indicates that the attribute may be updated by Consumers but are not retrievable by Consumers, typically for security reasons. Write-only attributes shall appear in the serialization of Resources sent to Providers but shall never appear in the serialization of Resources sent from Providers.

5.5 Data types and their serialization

Unless specifically asked to not include certain attributes in the Resource representation, the absence of an optional attribute in the representation means that the attribute has no value (i.e., is undefined), meaning there is no notion of an optional attribute having an implied value. Note that a client cannot distinguish (from just looking at the returned representation) whether a particular attribute is not supported from one that does not exist. Likewise, an absent attribute from a Resource representation as the input to an update operation means that the Consumer is requesting that the Provider remove that attribute.

The following clauses describe the data types and values that are used within the model definition tables.

5.5.1 boolean

A value as defined by xs:boolean per [XML Schema – Part 2](#), with the exception that the only allowable values are either "true" or "false." The value is case sensitive.

If serialized in JSON these values shall be of JSON type: *boolean*

If serialized in XML these values shall be of XML Schema type: *xs:boolean*

5.5.2 dateTime

A value as defined by xs:dateTime per [XML Schema – Part 2](#), which is consistent with DMTF DSP4004 and ISO 8601. The timestamp should preserve time zone information, i.e., include a local time component and an offset from UTC.

Any constraints on the specific ranges allowed for any particular attribute are specified by that attribute's definition or at runtime by the Provider by the way of the metadata discovery mechanisms defined by this specification.

For example, Monday, May 25, 2012, at 1:30:15 PM EST is represented as:

2012-05-25T13:30:15-05:00

If serialized in JSON these values shall be of JSON type: *string*

If serialized in XML these values shall be of XML Schema type: *xs:dateTime*

5.5.3 duration

A value as defined by xs:duration per [XML Schema – Part 2](#). Any constraints on the specific ranges allowed for any particular attribute shall be specified by that attribute's definition or at runtime by the Provider by the way of the metadata discovery mechanisms defined by this specification.

If serialized in JSON these values shall be of JSON type: *string*

If serialized in XML these values shall be of XML Schema type: *xs:duration*

1307 **5.5.4 integer**

1308 A value as defined by xs:integer per [XML Schema – Part 2](#). Any constraints on the specific ranges
 1309 allowed for any particular attribute shall be specified by that attribute's definition or at runtime by the
 1310 Provider by the way of the metadata discovery mechanisms defined by this specification.

1311 If serialized in JSON these values shall be of JSON type: *number*

1312 If serialized in XML these values shall be of XML Schema type: *xs:integer*

1313 **5.5.5 string**

1314 A value as defined by xs:string per [XML Schema – Part 2](#). Any constraints on this type for any particular
 1315 attribute shall be specified by that attribute's definition or at runtime by the Provider by the way of the
 1316 metadata discovery mechanisms defined by this specification.

1317 If serialized in JSON these values shall be of JSON type: *string*

1318 If serialized in XML these values shall be of XML Schema type: *xs:string*

1319 If serializing an attribute of type string, the serialization shall omit this attribute in case of an empty string.

1320 **5.5.6 ref**

1321 A reference to another Resource.

1322 References allow for Consumers to navigate to Resources. By starting at the Cloud Entry Point and
 1323 following the references that appear in the retrieved Resources, Consumers are able to recursively
 1324 discover and navigate to all other Resources.

1325 As a general rule, if an attribute is of type "ref", its value shall be held by an attribute named "href"
 1326 (both in JSON and XML).

1327 **JSON serialization:**

1328 In the JSON serialization the href property appears as of type "string." If an attribute is of type "ref",
 1329 the name of this attribute shall appear as a key, with the href property as it a nested value. For example,
 1330 a Resource attribute "myvolume" of type "ref" is serialized as:

1331

```
"myvolume": { "href": string }
```

1332 **XML serialization:**

1333 In the XML serialization the href attribute appears as type "xs:anyURI." If an attribute is of type "ref",
 1334 the name of this attribute shall appear as name of an XML element with the href property as an (XML)
 1335 attribute. For example, a Resource attribute "myvolume" of type "ref" is serialized as:

1336

```
<myvolume href="xs:anyURI"/>
```

1337 References in both JSON and XML have an extensibility point that allows for additional information (such
 1338 as the target Resource to be included "by value") if supported. For convenience, the JSON and XML
 1339 representations, as shown above, exclude the implicit extensibility points that would allow for the
 1340 attributes of the target Resource to be included if desired. So, more accurately the above representations
 1341 might be written as follows:

1342 For JSON:

1343

```
"myvolume": { "href": string, ... }
```

1344 and in XML:

```
<myvolume href="xs:anyURI"> xs:any* </myvolume>
```

However, for brevity the extensibility points are excluded from the serialization of the Resources.

5.5.7 map

A list of key/value pairs. The same "key" shall not be used more than once within an attribute. The "key" is case sensitive.

If serializing an attribute of type map, the serialization shall omit this attribute in case of an empty map.

5.5.8 structure

Attributes of this type are complex attributes made up of a set of nested attributes. For each attribute of this type, there is an additional table defining those nested attributes.

A nested structure can be considered a complex type definition. Structures may be named or unnamed. Table 2 is an example of named structure:

Table 2 – Named structure

Name	summary	
Attribute	Type	Description
low	number	Number of "low" occurrences
medium	number	Number of "medium" occurrences
high	number	Number of "high" occurrences
critical	number	Number of "critical" occurrences

JSON serialization:

In JSON, the name of the structure (i.e., of the type it represents) never appears. In other words, whether the structure is named or not does not matter. An attribute named "systemIncidents" of type "summary" (as above) is serialized as follows:

```
"systemIncidents": {
  "low": number,
  "medium": number,
  "high": number,
  "critical": number
}
```

XML serialization:

In XML, the name of the structure (i.e., of the type it represents) never appears. In other words, whether the structure is named or not does not matter. The same previous "systemIncidents" example is serialized so that the structure sub-attributes become XML attributes of a <systemIncidents> XML element wrapper:

```
<systemIncidents low="xs:integer" medium="xs:integer" high="xs:integer"
  critical="xs:integer"/>
```

NOTE A large number of sub-attributes of atomic type in a structure may be represented alternatively as XML child elements for better readability. Both options are available; however, the same structure shall be serialized the same way across Resources.

5.5.9 byte[]

An arbitrary set of bytes meant to represent a block of binary data. Any constraints on this type for any particular attribute shall be specified by that attribute's definition or at runtime by the Provider by the way of the metadata discovery mechanisms defined by this specification.

If serialized in JSON, these values shall be of JSON type: *string*

If serialized in XML, these values shall be of XML Schema type: *xs:hexBinary*

5.5.10 URI

The format and syntax of the attributes of type "URI" is defined by [RFC3986](#).

Unless otherwise noted, this specification does not mandate whether Providers use relative or absolute URI in the HTTP response bodies.

If URIs are specified as relative URIs, they shall be relative to the baseURI.

The algorithm used for converting a relative URI to an absolute URI shall be as described in section 5.2 of [RFC3986](#). Table 3 illustrates how relative URIs are resolved against base URIs:

Table 3 – Converting a relative URI to an absolute URI

Base URI	Relative URI	Absolute URI
http://example.com/	p1/file	http://example.com/p1/file
http://example.com/c1/	p1/file	http://example.com/c1/p1/file
http://example.com/c1/c2/	p1/file	http://example.com/c1/c2/p1/file

If relative URIs are used, the baseURI shall end with a trailing slash and relative URIs shall not begin with a leading slash. This format is consistent with most URI resolve utilities and produces the same results as a simple string concatenation algorithm.

If serialized in JSON, these values shall be of JSON type: *string*

If serialized in XML, these values shall be of XML Schema type: *xs:anyURI*

5.5.11 Arrays

An array represents an ordered list of items of the same type. An array shall always appear as an attribute of a Resource, and is only accessible as such (it is not a separately addressable Resource). If a Resource is deleted, the items in its arrays shall also be deleted. However, in case these items were just references to other Resources, these referred Resources are not affected. (See the semantics of references in 5.7.)

Attributes that are arrays are defined by using the notation *itemType[]*, where *itemType* is the type name for each item of the array. If the type is a structure, not a simple data type, it is recommended as a convention in the model that the name of an array be the plural of a name that characterizes each item. For example, an array of volume items or of references to these may be named "volumes."

If an attribute is of type of references (*ref[]*) – and more generally array of an atomic type – the definition in the model shall include an "Array item name" that may be used in its serialization.

JSON serialization:

Within this specification, arrays in JSON are serialized with a wrapper property. The wrapper name shall be same as the attribute name for the array. For example, a "things" attribute of type "thing[]" is serialized as:

```

"things" : [
  { ... }, +
] ?

```

If the items in the array are structures, the structure name shall not be present in the JSON serialization.

In the case of an array of references, i.e., where the "ref" type applies to each element of the array, each element shall simply be serialized as an href property within a JSON array. For example, an array "things" of type "ref[]" is serialized as:

```

"things": [
  { "href": string }, +
] ?

```

NOTE If serializing arrays, conformant implementations shall not include empty arrays (i.e., arrays that contain no child properties) in the JSON serialization. Notice that the child of the "things" property is defined with a "+", meaning at least one child is required. This requirement ensures that the JSON serialization is minimized and only includes the wrapping "things" element if, and only if, there is at least one "thing" in the array.

XML serialization:

The XML serialization of arrays requires each item of the array to be represented as an element. These elements shall be consecutive and contiguous in the serialization and the name of each element (tag name) shall be the name of the element type (the name that appears before "[]" in the array type). For example, a "things" attribute shall be serialized as a list of items named "thing", where "thing" is the name of a structure:

```

<thing>
  ...
</thing> *

```

There is no wrapper element for an array in XML.

In the case of an array of references, i.e., where the "ref" type applies to each element of the array, the array is serialized as a list of XML elements without wrapper. Each element is named per the "Array item name" value specified in the attribute's definition. For example, an array "things" of type "ref[]" where the "Array item name" is "thing" is serialized as:

```

<thing href="xs:anyURI"/> +

```

5.5.12 Collections

Like arrays, Collections are groupings of Resources of the same type. In contrast with arrays, Collections are themselves Resources that have their own URI and can be independently accessed. Collections also allow for an optimized and convenient interaction pattern by providing a specialized set of operations that avoid replacing a large number of items when updating the set.

This specification uses Collections if the set of items in the list is modified often and potentially by multiple Consumers. Conversely, arrays are used if it is expected that the list of items is not modified often or can be easily modified by substitution of the entire list, and thus the overhead of managing these items as separate Resources might be burdensome.

Attributes that are Collections are represented as type "collection[itemType]." The Resource type of the Collection items are specified inside the brackets; for example an attribute that is a Collection of Machines is expressed as "collection[Machine]." These are serialized as a reference to a Collection Resource. For brevity, while these attributes are "references" the word "ref" or "reference" does not appear in the model definition tables - simply the type "collection[itemType]" appears.

ISO/IEC 19831:2015 (E)

To each one of these Resource items, shall correspond an entry in the Collection. These Resources items are assumed to be of a complex type and are separately addressable and manageable. While different Collections contain entries of different Resource types, all Collections follow the pattern described below:

- Collections shall contain an `id` attribute that acts as a "self pointer." Retrieving the data at this reference shall return the Collection. In the XML representation, each Collection shall be wrapped by a `<Collection>` element.
- Collections shall contain a `count` attribute that indicates the number of Resources in the Collection at the time the Collection was queried.
- Collections shall contain a list of Resources that make up the Collection. As with all arrays, if there are no Resources in the Collection, the serialization of the list shall be omitted.
- As with all Resources in the CIMI model, each Resource in the Collection shall have an `id` attribute that acts as a "self pointer." Retrieving the data at this reference shall return just that one Resource and not any parent Resource, such as the Collection or array attribute.
- Adding new Resources to the Collection shall be done through the "add" operation defined within the Collection. Note that lack of an "add" operation on the Collection indicates that new Resources are not permitted at that time.
- Deleting Resources from the Collection shall be done through a "delete" operation on the Resource itself.
- Unless otherwise specified, deleting a Collection shall also delete all of the Resources that make up the Collection, but shall not delete any tertiary Resources referenced by the to-be-deleted Collection Resources.
- Collections shall be deleted if their owning Resource is deleted.

The Resources in a Collection are of two kinds:

- an infrastructure Resource (such as those listed in the Cloud Entry Point, or those embedded in an entity such as the `disks` inside a `Machine`)
- an intermediary Resource that holds a reference to an infrastructure Resource, called the "target Resource"

By convention, intermediary Resources have a name that concatenates the name of the Resource owning the Collection, with the name of the target Resource, e.g., `MachineVolume` is the name of the intermediary Resource that is used to connect a `Machine` to a `Volume`.

Collections of intermediary Resources allow for decoupling the lifecycle of a Collection (and of its owning entity) from the lifecycle of the actual target Resources. For example, deleting a Collection shall delete its intermediary Resources but not its target Resources. In case the reference to the target Resource is a mandatory attribute of the intermediary Resource, the intermediary Resource cannot have a longer lifecycle than the target Resource.

- If a target Resource is deleted, the Provider shall also delete any intermediary Resource that has a reference to this Resource as the value of a mandatory attribute.

The serialization of Collections shall adhere to the following pattern:

JSON serialization:

```
{ "resourceURI": string,  
  "id": string,
```

```

1497     "count": number,
1498     "resourceSpecificGroupingName": [
1499         { "resourceURI": string,
1500           "id": string,
1501           "name": string, ?
1502           "description": string, ?
1503           "created": string, ?
1504           "updated": string, ?
1505           "properties": { string: string, + }, ?
1506           ... entry specific data ...
1507           "operations": [
1508               { "rel": "edit", "href": string }, ?
1509               { "rel": "delete", "href": string } ?
1510           ] ?
1511           ...
1512         } +
1513     ], ?
1514     "operations": [ { "rel": "add", "href": string } ? ]
1515     ...
1516 }

```

XML serialization:

```

1518 <Collection resourceURI="xs:anyURI" xmlns="http://schemas.dmtf.org/cimi/1">
1519     <id> xs:anyURI </id>
1520     <count> xs:integer </count>
1521     <ResourceSpecificElementName>
1522         <id> xs:anyURI </id>
1523         <name> xs:string </name> ?
1524         <description> xs:string </description> ?
1525         <created> xs:dateTime </created> ?
1526         <updated> xs:dateTime </updated> ?
1527         <property key="xs:string"> xs:string </property> *
1528         ... entry specific data ...
1529         <operation rel="edit" href="xs:anyURI"/> ?
1530         <operation rel="delete" href="xs:anyURI"/> ?
1531         <xs:any>*
1532     </ResourceSpecificElementName> *
1533     <operation rel="add" href="xs:anyURI"/> ?
1534     <xs:any>*
1535 </Collection>

```

Where the resourceURI attributes shall contain the Collection or Resource specific URIs for that type of Collection, and resourceSpecificGroupingName and ResourceSpecificElementName shall be replaced with the name of the Collection-specific Resource name, e.g., machines in JSON or Machine in XML.

5.5.12.1 Adding items to Collections

Invoking the "add" operation of a Collection shall add a new Resource to the Collection. The contents of the request body shall be either a representation of the new Resource being added to the Collection, or a representation of the Template associated with the new Resource being created. This specification indicates which Resources require the use of a Template.

For example, to add a new Volume to the volumes Collection of a Machine, the "add" operation's request body shall be serialized as follows:

JSON serialization:

```
{ "resourceURI": "http://schemas.dmtf.org/cimi/1/MachineVolume",
  "initialLocation": string,
  "volume": { "href": string }
}
```

XML serialization:

```
<MachineVolume xmlns="http://schemas.dmtf.org/cimi/1">
  <initialLocation> xs:string </initialLocation>
  <volume href="xs:string"/>
</MachineVolume>
```

Note that while deleting this type of Resource from the Collection deletes and removes the Resource from the Collection, it shall not delete the referenced target Resource itself - in this case the Volume.

If creating a new Resource that requires the use of a Template, the "add" operation shall contain:

- The "common attributes" as defined by clause 5.10.1.
- The Resource specific data needed to create it. This data shall either be a reference to the Resource-specific Template Resource or the Resource-specific Template Resource itself inlined.
- In the XML case, a wrapper element (named after the pattern <ResourceNameCreate>).

For example, to create a new Machine (which requires the use of a Template) and add it to the MachineCollection, the "add" operation of the MachineCollection shall be serialized as follows:

JSON serialization:

```
{ "resourceURI": "http://schemas.dmtf.org/cimi/1/MachineCreate", ?
  "name": string, ?
  "description": string, ?
  "properties": { string: string, + }, ?
  "machineTemplate": { "href": string ?}
  ...
}
```

XML serialization:

```

<MachineCreate xmlns="http://schemas.dmtf.org/cimi/1">
  <name> xs:string </name> ?
  <description> xs:string </description> ?
  <property key="xs:string"> xs:string </property> *
  <machineTemplate href="xs:anyURI"? />
  <xs:any>*
</MachineCreate>

```

The MachineCollection has a new Machine:

JSON serialization:

```

{ "resourceURI": "http://schemas.dmtf.org/cimi/1/Machine",
  "id": string,
  "name": string,
  ...
}

```

XML serialization:

```

<Machine xmlns="http://schemas.dmtf.org/cimi/1">
  <id> xs:anyURI </id>
  <name> xs:string </name>
  ...
</Machine>

```

The processing of the "add" operation shall adhere to the semantics defined in clause 4.2.1.1.

Regardless of whether a Template is used, the "add" operation shall create the new Resource and add it to the Collection and a reference (URI) to the new entry shall be returned in the response message in the HTTP Location header.

5.5.13 "Any" type

Some attributes are polymorphic and can hold various data types, the list of which is indicated in their description. In such cases, the type of the attribute shall be indicated as "any" in the model representation.

5.5.14 Empty attribute values

Attributes of the following types are omitted in cases where they have an empty value: string, map, array, and Collection. Apart from being "Provider optional" or "Consumer optional", an empty value is the third reason that the serialization schema contains an '?' or an '*' for an attribute.

Other attribute types do not have empty values and shall not be omitted from the serialization for this reason.

5.6 Units

Some of the Resources defined by this specification have attributes that describe an amount of something that belongs to, or is associated with, that Resource. For example, the `Machine` Resource

has a **memory** attribute that describes "the size of the memory allocated to this machine." The allowable units of these attributes are listed in Table 4. Their meaning is defined in [IEC 80000-13:2008](#). Their numerical equivalents are provided here for convenience:

Table 4 – Numerical equivalents for attributes

String	Numerical Value	String	Numerical Value
kilobyte	10 ³	kibibyte	2 ¹⁰
megabyte	10 ⁶	mebibyte	2 ²⁰
gigabyte	10 ⁹	gibibyte	2 ³⁰
terabyte	10 ¹²	tebibyte	2 ⁴⁰
petabyte	10 ¹⁵	pebibyte	2 ⁵⁰
exabyte	10 ¹⁸	exbibyte	2 ⁶⁰
zettabyte	10 ²¹	zebibyte	2 ⁷⁰
yottabyte	10 ²⁴	yobibyte	2 ⁸⁰

5.7 Relationship semantics

A reference between two Resource instances has the semantics of a simple "association." In particular, unless specified otherwise, (a) the same referred instance can be referred by other Resource instances, i.e., be "shared," and (b) the referred Resource instance is not affected if deleting the referring Resource instance (i.e., the Delete operation is a "shallow delete" by default).

The embedding of a subresource inside another Resource, has the semantics of a "composition" (or whole-part relationship in UML). In particular, unless specified otherwise, (a) an embedded subresource cannot be shared by several Resource instances, and (b) if deleting an embedding Resource instance, the embedded subresource instances are also deleted.

5.8 Operations

All Resource operations defined by this specification are optional for Providers to support. Consumers, by the way of examination of a Resource's ResourceMetadata, can determine which operations are supported. However, even for those operations that are supported Consumers still need to examine each Resource's representation to determine which operations are supported at that moment. Whether an operation is supported is based on a number of factors, including state of the Resource and access control rights of the Consumer. Also see clause 4.2. Operations and states are coupled; i.e., if implementing a state-changing Resource operation defined in this specification, the corresponding state(s) shall also be implemented. See the Resource-specific "Operations" clauses for additional detail.

The "State" attribute of Resources that have this attribute shall only change value if

- an operation is performed on this Resource and this operation requires a state change, or
- an error occurred, in this case the "State" attribute shall obtain the value "ERROR".

For example, for a 'start' operation on a Machine both the STARTING and the STARTED states are required to be supported by the Machine, while the Machine can only leave the STARTED state after another state changing operation is requested, unless an error occurs.

Providers can define additional operations and states. Such extensions shall fall into one of these categories:

- A new operation that starts from a CIMI-defined state, or leads to a CIMI-defined state, or both. In the latter case, if a CIMI-defined operation already exists for this transition between two CIMI-defined states, it shall also be supported by the Provider in addition to the new operation.
- A new Resource state. In that case, a new operation that leads to that state shall also be created. In other words, a Provider-defined operation has to be performed before a Provider-defined state can be reached."

- c) A new operation that transitions between two Provider-defined states.

5.9 Alternative model formats

It is expected that this specification is implemented by using a variety of technologies. As a convenience, the definition of the model elements are provided in alternative formats that are easily consumable by technology-specific tooling.

This model is also available in a CIM/MOF format [CIMI-CIM].

In the event of inconsistencies between the various formats, the normative text within this specification takes precedence over the XML Schemas and alternative formats, which in turn take precedence over examples.

5.10 Resources

The following clauses detail the attributes of the Resources defined by the CIMI model.

5.10.1 Common attributes

Except for ResourceMetadata and Collection Resources (see 5.5.12), the Resources described by this document share the following common attributes; see Table 5. There are different requirements for primary and secondary CIMI resources. All Resources that are element types of Collections in the CloudEntryPoint shall be primary CIMI resources. All other Resources shall be secondary CIMI resources. An exception to this rule is that the CloudEntryPoint shall be considered a primary Resource.

For example, Machine is a primary CIMI resource as the CloudEntryPoint has a Collection with Machine as its element type. However, for example, MachineVolume is a secondary CIMI resource because the CloudEntryPoint does not have a Collection with MachineVolume as its element type.

Table 5 – Common attributes

Attribute	Type	Description
id	URI	The unique URI identifying this Resource; assigned upon Resource creation. This attribute value shall be unique in the Provider's cloud. Constraints for primary and secondary Resources: Provider: support mandatory; immutable Consumer: support mandatory; read-only
name	string	The human-readable name of this Resource; assigned by the creator as a part of the Resource creation input. Constraints for primary Resources: Provider: support mandatory; mutable Consumer: support optional; read-write Constraints for secondary Resources: Provider: support optional; mutable Consumer: support optional; read-write
description	string	The human-readable description of this Resource; assigned by the creator as a part of the Resource creation input. Constraints for primary Resources: Provider: support mandatory; mutable Consumer: support optional; read-write Constraints for secondary Resources: Provider: support optional; mutable Consumer: support optional; read-write
created	dateTime	The timestamp when this Resource was created. The format should be unambiguous, and the value is immutable .

Attribute	Type	Description															
		Constraints for primary and secondary Resources: Provider: support optional; immutable Consumer: support optional; read-only															
updated	<i>dateTime</i>	The time at which the last explicit attribute update was made on the Resource. Note, while operations, such as "stop", do implicitly modify the 'state' attribute, they do not change the 'updated_time'. Constraints for primary and secondary Resources: Provider: support optional; mutable Consumer: support optional; read-only															
properties	<i>map</i>	A map of key/value pairs (each entry called a "property"), some of which may control one or more aspects this Resource. Properties may also serve as an extension point, allowing Consumers to record additional information about the Resource. The same "key" shall not be used more than once within a "properties" attribute. Each property shall contain the following nested data: <table border="1"> <tr> <th colspan="3">Name</th></tr> <tr> <td colspan="3"><i>property</i></td></tr> <tr> <th>Data</th><th>Type</th><th>Description</th></tr> <tr> <td>key</td><td><i>string</i></td><td> The name of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write </td></tr> <tr> <td>value</td><td><i>string</i></td><td> The value of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write </td></tr> </table> Constraints for primary Resources: Provider: support mandatory; mutable Consumer: support optional; read-write Constraints for secondary Resources: Provider: support optional; mutable Consumer: support optional; read-write	Name			<i>property</i>			Data	Type	Description	key	<i>string</i>	The name of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write	value	<i>string</i>	The value of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
Name																	
<i>property</i>																	
Data	Type	Description															
key	<i>string</i>	The name of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write															
value	<i>string</i>	The value of the property. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write															

1673 The following pseudo-schemas describe the serialization of these attributes in both JSON and XML:

1674 **JSON serialization:**

```

1675 "id": string,
1676 "name": string, ?
1677 "description": string, ?
1678 "created": string, ?
1679 "updated": string, ?
1680 "properties": { string: string, + }, ?

```

1681 **XML serialization:**

```

1682 <id> xs:anyURI </id>
1683 <name> xs:string </name> ?
1684 <description> xs:string </description> ?
1685 <created> xs:dateTime </created> ?
1686 <updated> xs:dateTime </updated> ?
1687 <property key="xs:string"> xs:string </property> *

```

5.11 Resource metadata

Implementations of this specification should allow for Consumers to discover the metadata associated with each supported Resource type. Doing so allows for the discovery of Provider-defined constraints on the CIMI defined attributes as well as discovery of any new extension attributes or operations that the Provider may have defined. A ResourceMetadata instance contains metadata describing a particular Resource type – e.g., Network, or Machine – including any Provider-specific capabilities or features. The mechanism by which this metadata is made available is protocol specific.

Note that while this specification declares the ResourceMetadata as mutable attributes, it is expected that only administrative users associated with the Provider will update them. Consequently they remain read-only for Consumers.

Each Resource's metadata shall contain the following pieces of information:

Table 6 – ResourceMetadata attributes

Name	ResourceMetadata																			
Type URI	http://schemas.dmtf.org/cimi/1/ResourceMetadata																			
Attribute	Type	Description																		
id	URI	The unique URI identifying this Resource; assigned upon Resource creation. This attribute value is immutable , and shall be unique in the Provider's cloud. Constraints: Provider: support mandatory; immutable Consumer: support mandatory; read-only																		
typeURI	URI	A unique URI associated with, and denoting, the described Resource type. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
name	string	The name of the described Resource type. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
attributes	attribute[]	A set of Provider-defined metadata that can be used by clients to discover any metadata associated with each attribute of the described Resource type, including the set of extension attributes not defined in this specification. Each attribute shall contain the following nested data: <table> <tr> <th>Name</th><th colspan="2">attribute</th></tr> <tr> <th>Data</th><th>Type</th><th>Description</th></tr> <tr> <td>name</td><td>string</td><td>The name of the attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write</td></tr> <tr> <td>namespace</td><td>URI</td><td>The namespace in which this attribute is defined. It is recommended that a dereference of this URI returns information about the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write</td></tr> <tr> <td>type</td><td>string</td><td>The data type of the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI-defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write</td></tr> <tr> <td>required</td><td>boolean</td><td>Indicates whether this Resource requires this attribute</td></tr> </table>	Name	attribute		Data	Type	Description	name	string	The name of the attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write	namespace	URI	The namespace in which this attribute is defined. It is recommended that a dereference of this URI returns information about the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write	type	string	The data type of the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI-defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write	required	boolean	Indicates whether this Resource requires this attribute
Name	attribute																			
Data	Type	Description																		
name	string	The name of the attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
namespace	URI	The namespace in which this attribute is defined. It is recommended that a dereference of this URI returns information about the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
type	string	The data type of the attribute. This shall not be present if describing a CIMI-defined attribute, but shall be present if describing a non-CIMI-defined attribute. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
required	boolean	Indicates whether this Resource requires this attribute																		

Name	ResourceMetadata																				
Type URI	http://schemas.dmtf.org/cimi/1/ResourceMetadata																				
Attribute	Type	Description																			
			to be present. If absent the implied value is "false." Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
		value constraints	any Type-specific data that describes any constraints on values of this attribute. If absent, there are no constraints. Note that the serialization of these "value constraints" shall be determined by the type of the attribute; see clause 5.11.1. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																		
		Constraints: Provider: support optional; mutable Consumer: support optional; read-write																			
capabilities	capability[]	A set of Provider-defined metadata that can be used by Consumer to discover any capability or feature provided by this Provider. Each capability shall contain the following nested data: <table><tr><td>Name</td><td colspan="2">capability</td></tr><tr><td>Data</td><td>Type</td><td>Description</td></tr><tr><td>name</td><td>string</td><td>The name of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write</td></tr><tr><td>uri</td><td>URI</td><td>A URI that uniquely identifies the capability at a global level. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write</td></tr><tr><td>description</td><td>string</td><td>The human-readable description of the semantic of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write</td></tr><tr><td>value</td><td>any</td><td>The value of the capability. The specific type varies depending on the definition of the capability. If not present the capability defaults to a "boolean" type with a value of "true" indicating that the specific capability is supported by the Provider. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write</td></tr></table> Constraints: Provider: support optional; mutable Consumer: support optional; read-write		Name	capability		Data	Type	Description	name	string	The name of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write	uri	URI	A URI that uniquely identifies the capability at a global level. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write	description	string	The human-readable description of the semantic of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write	value	any	The value of the capability. The specific type varies depending on the definition of the capability. If not present the capability defaults to a "boolean" type with a value of "true" indicating that the specific capability is supported by the Provider. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
Name	capability																				
Data	Type	Description																			
name	string	The name of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write																			
uri	URI	A URI that uniquely identifies the capability at a global level. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																			
description	string	The human-readable description of the semantic of the capability. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write																			
value	any	The value of the capability. The specific type varies depending on the definition of the capability. If not present the capability defaults to a "boolean" type with a value of "true" indicating that the specific capability is supported by the Provider. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write																			
actions	action[]	A set of Provider defined operations that can be used by consumers to act on the Resource. This set represents all operations defined for this described Resource type, which may be a superset of those operations a particular Consumer is actually allowed to use. The subset of allowed operations for a particular Consumer shall be those operations returned to this Consumer if querying an instance of the described Resource type. Note that this attribute is called "actions" so as not to conflict with the ResourceMetadata Resource's own operations. Each operation shall contain the following nested data: <table><tr><td>Name</td><td colspan="2">action</td></tr><tr><td>Data</td><td>Type</td><td>Description</td></tr><tr><td>name</td><td>string</td><td>The name of the operation.</td></tr></table>		Name	action		Data	Type	Description	name	string	The name of the operation.									
Name	action																				
Data	Type	Description																			
name	string	The name of the operation.																			

Name	ResourceMetadata		
Type URI	http://schemas.dmtf.org/cimi/1/ResourceMetadata		
Attribute	Type	Description	
			Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
	uri	URI	A URI that uniquely identifies the operation at a global level. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
	description	string	The human-readable description of the semantic of the operation. Constraints: Provider: support mandatory; mutable Consumer: support optional; read-write
	method	string	The protocol-dependent verb to use to perform the operation. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
	inputMessage	string	The body mimeType of the request message; it may depend on the model format chosen by the Provider. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
	outputMessage	string	The body mimeType of the response message; it may depend on the model format chosen by the Provider. Constraints: Provider: support mandatory; mutable Consumer: support mandatory; read-write
	Constraints: Provider: support optional; mutable Consumer: support optional; read-write		

1700 When implementing or using ResourceMetadata, Providers and Consumers shall adhere to the
1701 syntax and semantics of its attributes as described in the above table as well as in the tables describing
1702 embedded Resources or related Collections. Both Consumer and Provider shall serialize this Resource
1703 as described below. The following pseudo-schemas (see notation in 1.3) describe the serialization of the
1704 Resource in both JSON and XML:

1705 **JSON media type:** application/json

1706 **JSON serialization:**

```

1707 { "resourceURI": "http://schemas.dmtf.org/cimi/1/ResourceMetadata",
1708   "id": string,
1709   "typeURI": string,
1710   "name": string,
1711   "attributes" : [
1712     { "name": string,
1713       "namespace": string, ?
1714       "type": string, ?
1715       "required": boolean, ?
1716       ...value constraints...? } *

```

```

1717 ], ?
1718 "capabilities": [
1719   { "name": string, ?
1720     "uri": string,
1721     "description": string, ?
1722     "value": any } *
1723 ], ?
1724 "actions" : [
1725   { "name": string,
1726     "uri": string,
1727     "description": string, ?
1728     "method": string,
1729     "inputMessage": string, ?
1730     "outputMessage": string ? }, *
1731 ], ?
1732 "operations": [
1733   { "rel": "edit", "href": string }, ?
1734   { "rel": "delete", "href": string } ?
1735 ] ?
1736 ...
1737 }

```

1738 **XML media type:** application/xml

1739 **XML serialization:**

```

1740 <ResourceMetadata xmlns="http://schemas.dmtf.org/cimi/1">
1741   <id> xs:anyURI </id>
1742   <name> xs:string </name>
1743   <typeURI> xs:anyURI </typeURI>
1744   <attribute name="xs:string" namespace="xs:anyURI"? type="xs:string"
1745     required="xs:boolean"? >
1746     ...value constraints...?
1747   </attribute> *
1748   <capability name="xs:string"? uri="xs:anyURI" description="xs:string"?>
1749     xs:any*
1750   </capability> *
1751   <action name="xs:string" uri="xs:anyURI" description="xs:string"?
1752     method="xs:string" inputMessage="xs:string"?
1753     outputMessage="xs:string"? /> *
1754   <operation rel="edit" href="xs:anyURI"/> ?
1755   <operation rel="delete" href="xs:anyURI"/> ?

```

1756 <xs:any>*

1757 </ResourceMetadata>

1758 Additional metadata about the Resource or attributes may be included by the Provider.

1759 **5.11.1 Serialization of attribute value constraints**

1760 The following examples describe the values, syntax, and serialization of the "value constraints" attribute
 1761 (sub-attribute of "attributes"), which has a type of "any."

1762 **type="string"**

1763 The JSON shall be of the form:

1764 "values": [*string*, +] ?

1765 The XML shall be of the form:

1766 <value> *xs:string* </value> *

1767 **type="integer"**

1768 The JSON shall be of the form:

1769 "values": [*number*, +], ?

1770 "ranges": [{ "low": *number*, "high": *number* }, +] ?

1771 The XML shall be of the form:

1772 <value> *xs:integer* </value> *

1773 <range low="*xs:integer*" high="*xs:integer*" /> *

1774 The total value space of an 'integer' attribute is the accumulation of all values and ranges.

1775 **type="boolean"**

1776 The JSON shall be of the form:

1777 "value": *boolean* ?

1778 The XML shall be of the form:

1779 <value> *xs:boolean* </value> ?

1780 Only one "value" is permitted. It indicates whether the attribute is required to be either "true" or "false".

1781 **5.11.1.1 Examples**

1782 The following example shows a sample metadata document for a VolumeConfiguration Resource
 1783 in XML that lists the allowable values for the "format" attribute and has been extended with a "Location"
 1784 string attribute:

1785 <ResourceMetadata xmlns="http://schemas.dmtf.org/cimi/1">

1786 <id> http://example.org/types/VC </id>

1787 <typeURI> http://schemas.dmtf.org/cimi/1/VolumeConfiguration </typeURI>

1788 <name> VolumeConfiguration </name>

1789 <attribute name="format" type="string" required="false">

1790 <value> ext4 </value>

1791 <value> ntfs </value>

```

1792     </attribute>
1793     <attribute name="Location" namespace="http://example.org/" type="string"/>
1794 </ResourceMetadata>

```

The following example shows the same VolumeConfiguration, but the "Location" attribute is restricted to a set of values and is required:

```

1797 <ResourceMetadata xmlns="http://schemas.dmtf.org/cimi/1">
1798   <id> http://example.org/types/VC </id>
1799   <typeURI> http://schemas.dmtf.org/cimi/1/VolumeConfiguration </typeURI>
1800   <name> VolumeConfiguration </name>
1801   <attribute name="format" type="string" required="false">
1802     <value> ext4 </value>
1803     <value> ntfs </value>
1804   </attribute>
1805   <attribute name="Location" namespace="http://example.org/" type="string"
1806     required="true">
1807     <value> NYC </value>
1808     <value> LAX </value>
1809   </attribute>
1810 </ResourceMetadata>

```

The following example shows the same VolumeConfiguration serialized in JSON:

```

1811 { "resourceURI": "http://schemas.dmtf.org/cimi/1/VolumeConfiguration",
1812   "id": "http://example.org/types/VC",
1813   "typeURI": "http://schemas.dmtf.org/cimi/1/VolumeConfiguration",
1814   "name": "VolumeConfiguration",
1815   "attributes": [
1816     { "name": "format",
1817       "type": "string",
1818       "required": false,
1819       "values": [ "ext4", "ntfs" ]
1820     },
1821     { "name": "Location",
1822       "namespace": "http://example.org",
1823       "type": "string",
1824       "required": true,
1825       "values": [ "NYC", "LAX" ]
1826     }
1827   ]
1828 }
1829

```