CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 18, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks

July 17, 2017

CoAP Management Interface
draft-ietf-core-comi-01

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access datastore and data node resources specified in YANG, or SMIv2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 18, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to [RFC8040] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data models specified in a standardized language, such as YANG, promotes interoperability between devices and applications from different manufacturers.

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small messges, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in the YANG data modelling language [RFC7950]: action, anydata, anyxml, client, configuration data, container, data model, data node, datastore, identity, instance identifier, key, key leaf, leaf, leaf-list, list, module, RPC, schema node, server, state data, submodule.

The following term is defined in [I-D.ietf-core-yang-cbor]: YANG schema item identifier (SID).

The following terms are defined in the CoAP protocol [RFC7252]: Confirmable Message, Content-Format.

The following terms are defined in this document:

data node resource: a CoAP resource that models a YANG data node.

datastore resource: a CoAP resource that models a YANG datastore.

event stream resource: a CoAP resource used by clients to observe YANG notifications.

target resource: the resource that is associated with a particular CoAP request, identified by the request URI.

data node instance: An instance of a data node specified in a YANG module and stored in the server.

notification instance: An instance of a schema node of type notification, specified in a YANG module implemented by the server. The instance is generated in the server at the occurrence of the corresponding event and reported by an event stream.

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data

nodes defined at the root of a YANG module or data nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance identifier: List instance identifier or single instance identifier.

data node value: The value assigned to a data node instance. Data node values are serialized into the payload according to the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for reading and modifying the content of datastore(s) used for the management of the instrumented node.

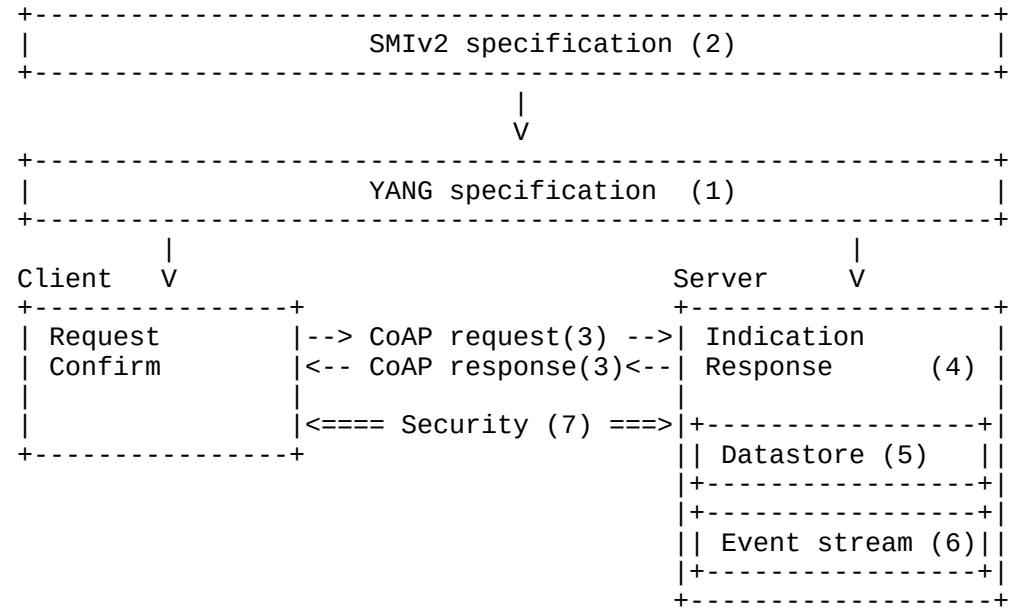

Figure 1: Abstract CoMI architecture

Figure 1 is a high-level representation of the main elements of the CoMI management architecture. The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request/response messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Request, Indication, Response, Confirm: The processes performed by the CoMI clients and servers.

(5) Datastore: A resource used to access configuration data, state data, RPCs and actions. A CoMI server may support multiple datastores to support more complex operations such as configuration rollback, scheduled update.

(6) Event stream: An observable resource used to get real time notifications. A CoMI server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.

(7) Security: The server MUST prevent unauthorized users from reading or writing any CoMI resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI is a RESTful protocol for small devices where saving bytes to transport counts. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CoMI is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF. Some major differences are cited below:

- CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats.

- CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

- CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

- CoMI does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.

- CoMI does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.

- CoMI and RESTCONF also differ in the handling of:

  * notifications.

  * default values.

2.2. Compression of YANG identifiers

In the YANG specification, items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric identifiers are used instead of these strings. YANG Schema Item iDentifier (SID) is defined in [I-D.ietf-core-yang-cbor] section 2.1.

When used in a URI, SIDs are encoded in based64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed.

```
SID in basae64 = URLsafeChar[SID >> 60 & 0x3F] |
                 URLsafeChar[SID >> 54 & 0x3F] |
                 URLsafeChar[SID >> 48 & 0x3F] |
                 URLsafeChar[SID >> 42 & 0x3F] |
                 URLsafeChar[SID >> 36 & 0x3F] |
                 URLsafeChar[SID >> 30 & 0x3F] |
                 URLsafeChar[SID >> 24 & 0x3F] |
                 URLsafeChar[SID >> 18 & 0x3F] |
                 URLsafeChar[SID >> 12 & 0x3F] |
                 URLsafeChar[SID >> 6 & 0x3F] |
                 URLsafeChar[SID & 0x3F]
```

For example, SID 1717 is encoded as follow.

```
URLsafeChar[1717 >> 60 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 54 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 48 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 42 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 36 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 30 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 24 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 18 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 12 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1717 >> 6 & 0x3F]  = URLsafeChar[26] = 'a'
URLsafeChar[1717 & 0x3F]       = URLsafeChar[53] = '1'
```

The resulting base64 representation of SID 1717 is "a1"

2.3. Instance identifier

Instance identifiers are used to uniquely identify data node instances within a datastore. This YANG built-in type is defined in [RFC7950] section 9.13. An instance identifier is composed of the data node identifier (i.e. a SID) and for data nodes within list(s) the keys used to index within these list(s).

When part of a payload, instance identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' URI-Query as defined in Section 5.1.

2.4. CBOR ordered map schematic

An ordered map is used as a root container of the application/yang-tree+cbor Content-Format. This datatype share the same functionalities as a CBOR map without the following limitations:

- The ordering of the pairs of data items is preserved from serialization to deserialization.
- Duplicate keys are allowed

This schematic is constructed using a CBOR array comprising pairs of data items, each pair consisting of a key that is immediately followed by a value. Unlike a CBOR map for which the length denotes the number of pairs, the length of the ordered map denotes the number of items (i.e. number of keys plus number of values).

The use of this schematic can be inferred from its context or by the presence of a preceding tag. The tag assigned to the Ordered map is defined in Section 11.4.

In the case of CoMI, the use of the ordered map as the root container of the application/yang-tree+cbor Content-Format is inferred, the Ordered map tag is not used.

2.5. Content-Formats

ComI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor]. All Content-Formats defined hereafter are constructed using one or both of these constructs:

- YANG data node value, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 4.

- YANG instance identifier, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1.

The following Content-formats are defined:

application/yang-value+cbor: represents a CBOR YANG document containing one YANG data node value. The YANG data node instance can be a leaf, a container, a list, a list instance, a RPC input, a RPC output, an action input, an action output, a leaf-list, an anydata or an anyxml. The CBOR encoding for each of these YANG data node instances are defined in [I-D.ietf-core-yang-cbor] section 4.

FORMAT: data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance, a RPC input, a RPC output, an action input, an actions output and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the URI carried in the associated request (i.e. GET, PUT, POST).

application/yang-values+cbor: represents a YANG document containing a list of data node values.

FORMAT: CBOR array of data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the corresponding instance-identifier carried in the FETCH request.

application/yang-tree+cbor: represents a CBOR YANG document containing a YANG data tree.

FORMAT: ordered map of single-instance-identifier, data-node-value

DELTA ENCODING: The SID part of the first instance-identifier within the ordered map is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-selectors+cbor: represents a CBOR YANG document containing a list of data node selectors (i.e. instance identifier).

FORMAT: CBOR array of instance-identifier

DELTA ENCODING: The SID part of the first instance-identifier within the CBOR array is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

application/yang-patch+cbor: represents a CBOR YANG document containing a list of data nodes to be replaced, created, or deleted.

For each data node instance, D, for which the instance identifier is the same as for a data node instance, I, in the targeted resource: the data node value of D replaces the data node value of I. When the data node value of D is null, the data node instance I is removed. When the targeted resource does not contain a data node instance with the same instance identifier as D, a new data node instance is created in the targeted resource with the same instance identifier and data node value as D.

FORMAT: CBOR array of instance-identifier, data-node-value

DELTA ENCODING: Same as Content-Format application/yang-tree+cbor

The different Content-formats usage is summarized in the table below:

Method	Resource	Content-Format
GET response	data node	/application/yang-value+cbor
PUT request	data node	/application/yang-value+cbor
POST request	data node	/application/yang-value+cbor
DELETE	data node	n/a
GET response	datastore	/application/yang-tree+cbor
PUT request	datastore	/application/yang-tree+cbor
POST request	datastore	/application/yang-tree+cbor
FETCH request	datastore	/application/yang-selectors+cbor
FETCH response	datastore	/application/yang-values+cbor
iPATCH request	datastore	/application/yang-patch+cbor
GET response	event stream	/application/yang-tree+cbor
POST request	rpc, action	/application/yang-value+cbor
POST response	rpc, action	/application/yang-value+cbor

3. Example syntax

This section presents the notation used for the examples. The YANG modules that are used throughout this document are shown in Appendix C. The example modules are copied from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

CBOR is used to encode CoMI request and response payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

SIDs in URIs are represented as a base64 number, SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Collection Interface. CoMI end-points that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.datastore, with example path: /c, where c is short-hand for CoMI. The path /c is recommended but not compulsory (see Section 8).

Three CoMI resources are accessible with the following three example paths:

/c: Datastore resource with path "/c" and using CBOR content encoding format. Sub-resouces of format /c/instance-identifier may be available to access directly each data node resource for this datastore.

/mod.uri: URI identifying the location of the YANG module library used by this server, with path "/mod.uri" and Content-Format "text/plain; charset=utf-8". An ETag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/s: Event stream resource to which YANG notification instances are reported. Notification support is optional, so this resource will not exist if the server does not support any notifications.

The mapping of YANG data node instances to CoMI resources is as follows. Every data node of the YANG modules loaded in the CoMI server represents a sub-resource of the datastore resource (e.g. /c/instance-identifier).

When multiple instances of a list exist, instance selection is possible as described in Section 5.1, Section 5.2.4, and Section 5.2.3.1.

The description of the management collection interface, with if=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

Function	Recommended path	rt
Datastore	/c	core.c.datastore
Data node	/c/instance-identifier	core.c.datanode
YANG module library	/mod.uri	core.c.moduri
Event steam	/s	core.c.eventstream

The path values are example values. On discovery, the server makes the actual path values known for these four resources.

5. CoMI Collection Interface

The CoMI Collection Interface provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data node resource
FETCH	Retrieve specific data nodes within a datastore resource
POST	Create a datastore resource or a data node resource, invoke an RPC or action
PUT	Create or replace a datastore resource or a data node resource
iPATCH	Idem-potently create, replace, and delete data node resource(s) within a datastore resource
DELETE	Delete a datastore resource or a data node resource

There is one Uri-Query option for the GET, PUT, POST, and DELETE methods.

Uri-Query option	Description
k	Select an instance within YANG list(s)

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

5.1. Using the 'k' Uri-Query option

The "k" (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a data node, and key1, key2 are the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

- For a given list, if a parent data node is a list, generate the keys for the parent list first.
- For a given list, generate key values in the order specified in the YANG module.

Key values are encoded using the rules defined in the following table.

YANG datatype	Uri-Query text content
uint8,uint16,unit32, uint64	int2str(key)
int8, int16,int32, int64	urlSafeBase64(CBORencode(key))
decimal64	urlSafeBase64(CBOR key)
string	key
boolean	"0" or "1"
enumeration	int2str(key)
bits	urlSafeBase64(CBORencode(key))
binary	urlSafeBase64(key)
identityref	int2str(key)
union	urlSafeBase64(CBORencode(key))
instance-identifier	urlSafeBase64(CBORencode(key))

In this table:

- The method int2str() is used to convert an integer value to a string. For example, int2str(0x0123) return the string "291".
- The method urlSafeBase64() is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. For example, urlSafeBase64(\xF9\x56\xA1\x3C) return the string "-VahPA".
- The method CBORencode() is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 5, item 8.

The resulting key string is encoded in a Uri-Query as specified in [RFC7252] section 6.5.

5.2. Data Retrieval

One or more data nodes can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [RFC8132].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] may be used, as explained in more detail in Section 7.

There are two additional Uri-Query options for the GET and FETCH methods.

Uri-Query option	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

5.2.1. Using the 'c' Uri-Query option

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data node resources. A 4.02 (Bad Option) error is returned if used for other methods or resource types.

If this Uri-Query option is not present, the default value is "a".

5.2.2. Using the 'd' Uri-Query option

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported. Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value by any client yet, the server MUST return the default value of the leaf.

If the target of a GET method is a data node that represents a container or list that has child resources with default values, and these have not been given value yet,

The server MUST not return the child resource if d= 't'

The server MUST return the child resource if d= 'a'.

If this Uri-Query option is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. An instance identifier is specified in the URI path prefixed with the example path /c.

FORMAT:

```
GET /c/instance-identifier

2.05 Content (Content-Format: application/yang-value+cbor)
data-node-value
```

The returned payload contains the CBOR encoding of the specified data node instance value.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix C.1, a request is sent to retrieve the value of system-state/clock/current-datetime specified in container system-state. The SID of system-state/clock/current-datetime is 1719, encoded in octal 3267, yields two 6 bit decimal numbers 26 and 55, encoded in base64, (according to table 2 of [RFC4648]) yields a3. The response to the request returns

the CBOR encoding of this leaf of type 'string' as defined in [I-D.ietf-core-yang-cbor] section 5.4.

REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"2014-10-26T12:16:31Z"

The next example represents the retrieval of a YANG container. In this case, the CoMI client performs a GET request on the clock container (SID = 1717; base64: a1). The container returned is encoded using a CBOR map as specified by [I-D.ietf-core-yang-cbor] section 4.2.

REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/yang-value+cbor)

```
{
  +2 : "2014-10-26T12:16:51Z",   / SID 1719 /
  +1 : "2014-10-21T03:00:00Z"    / SID 1718 /
}
```

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/yang-value+cbor)

```
[
  {
    +4 : "eth0",                / name  (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : true                   / enabled ( SID 1535) /
  },
  {
    +4 : "eth1",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type, (SID 1538) identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : false                  / enabled /
  }
]
```

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID=1534, base64: X-) within the interface list corresponding to the list key "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 5.4.

REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"Ethernet adaptor"

5.2.4. FETCH

The FETCH is used to retrieve multiple data node values. The FETCH request payload contains a list of instance-identifier encoded based on the rules defined by Content-Format application/yang-selectors+cbor in Section 2.5. The return response payload contains a list of values encoded based on the rules defined by Content-Format application/yang-values+cbor in Section 2.5. A value MUST be returned for each instance-identifier specified in the request. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

FORMAT:
  FETCH /c (Content-Format :application/yang-selectors+cbor)
  CBOR array of instance-identifier

  2.05 Content (Content-Format: application/yang-values+cbor)
  CBOR array of data-node-value

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix C.1. In the following example the value of current-datetime (SID 1719 and the interface list (SID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c (Content-Format :application/yang-selectors+cbor)
[
  1719,            / SID 1719 /
  [-186, "eth0"]   / SID 1533 with name = "eth0" /
]

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
[
  "2014-10-26T12:16:31Z",
  {
    +4 : "eth0",                / name (SID 1537) /
    +1 : "Ethernet adaptor",    / description (SID 1534) /
    +5 : 1179,                  / type (SID 1538), identity /
                                / ethernetCsmacd (SID 1179) /
    +2 : true                   / enabled (SID 1535) /
  }
]
```

5.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as CBOR arrays so messages will preserve their order.

5.3.2. POST

The CoAP POST operation is used in CoMI for creation of data node resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create a data node resource is sent with a confirmable CoAP POST message. The URI specifies the data node to be instantiated at the exception of list intances. In this case, for compactness, the URI specifies the list for which an instance is created.

FORMAT:

```
  POST /c/<instance identifier>
  (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

If the data node resource already exists, then the POST request MUST fail and a "4.09 Conflict" response code MUST be returned

5.3.2.1. Post example

The example uses the interface list from Appendix C.1. Example is creating a new list instance within the interface list (SID = 1533):

```
REQ: POST /c/X9 (Content-Format :application/yang-value+cbor)
{
  +4 : "eth5",              / name (SID 1537) /
  +1 : "Ethernet adaptor",  / description (SID 1534) /
  +5 : 1179,                / type (SID 1538), identity /
                            / ethernetCsmacd (SID 1179) /
  +2 : true                 / enabled (SID 1535) /
}


RES: 2.01 Created
```

5.3.3. PUT

A data node resource instance is created or replaced with the PUT method. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

```
FORMAT:
  PUT /c/<instanceidentifier>
           (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

5.3.3.1. PUT example

The example uses the interface list from Appendix C.1. Example is renewing an instance of the list interface (SID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0"
(Content-Format :application/yang-value+cbor)
{
  +4 : "eth0",             / name (SID 1537) /
  +1 : "Ethernet adaptor", / description (SID 1534) /
  +5 : 1179,               / type (SID 1538), identity /
                           / ethernetCsmacd ( SID 1179) /
  +2 : true                / enabled (SID 1535) /
}

RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent iPATCH method [RFC8132]. A request is sent with a confirmable CoAP iPATCH message.

There are no Uri-Query options for the iPATCH method.

The processing of the iPATCH command is specified by Content-Format application/yang-patch+cbor. In summary, if the CBOR patch payload contains a data node instance that is not present in the target, this instance is added. If the target contains the specified instance, the content of this instance is replaced with the value of the payload. A null value indicates the removal of an existing data node instance.

FORMAT:

```
  iPATCH /c (Content-Format :application/yang-patch+cbor)
  ordered map of instance-identifier, data-node-value

  2.04 Changed
```

5.3.4.1. iPATCH example

In this example, a CoMI client requests the following operations:

- Set "/system/ntp/enabled" (SID 1751) to true.
- Remove the server "tac.nrc.ca" from the"/system/ntp/server" (SID 1752) list.
- Add the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1752).

```
REQ: iPATCH /c (Content-Format :application/yang-patch+cbor)
[
  1751 , true,                          / enabled (1751) /
  [+1, "tac.nrc.ca"], null,             / server (SID 1752) /
  +0,                                   / server (SID 1752) /
    {
      +3 : "tic.nrc.ca",                / name (SID 1755) /
      +4 : true,                        / prefer (SID 1756) /
      +5 : {                            / udp (SID 1757) /
        +1 : "132.246.11.231"           / address (SID 1758) /
      }
    }
]

RES: 2.04 Changed
```

5.3.5. DELETE

A data node resource is deleted with the DELETE method.

```
FORMAT:
  Delete /c/<instance identifier>

  2.02 Deleted
```

5.3.5.1. DELETE example

The example uses the interface list from Appendix C.3. Example is deleting an instance of the interface list (SID = 1533):

```
REQ:   DELETE /c/X9?k="eth0"

RES:   2.02 Deleted
```

5.4. Full datastore access

The methods GET, PUT, POST, and DELETE can be used to request, replace, create, and delete a whole datastore respectively.

```
FORMAT:
  GET /c

  2.05 Content (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value
```

```
FORMAT:
  PUT /c (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value

  2.04 Changed

FORMAT:
  POST /c (Content-Format: application/yang-tree+cbor)
  ordered map of single-instance-identifier, data-node-value

  2.01 Created

FORMAT:
   DELETE /c

  2.02 Deleted
```

The content of the ordered map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request. When an Ordered map is used to carry a whole datastore, all data nodes MUST be identified using single instance identifiers (i.e. a SID), list instance identifiers are not allowed.

5.4.1. Full datastore examples

The example uses the interface list and the clock container from Appendix C.3. Assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the list interface (SID 1533) with one instance and the container Clock (SID 1717). After invocation of GET, a map with these two modules is returned:

```
REQ:  GET /c

RES: 2.05 Content (Content-Format :application/yang-tree+cbor)
[
  1717,                           / Clock (SID 1717) /
    {
      +2: "2016-10-26T12:16:31Z", / current-datetime (SID 1719) /
      +1: "2014-10-05T09:00:00Z"  / boot-datetime (SID 1718) /
    },
  -186,                            / clock (SID 1533) /
    {
      +4 : "eth0",                / name (SID 1537) /
      +1 : "Ethernet adaptor",    / description (SID 1534) /
      +5 : 1179,                  / type (SID 1538), identity: /
                                  / ethernetCsmacd (SID 1179) /
      +2 : true                   / enabled (SID 1535) /
    }
]
```

5.5. Event stream

Event notification is an essential function for the management of servers. CoMI allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The recommended path of the default event stream is /s. The server MAY support additional event stream resources to address different notification needs.

Reception of notification instances is enabled with the CoAP Observe [RFC7641] function. Clients subscribe to the notifications by sending a GET request with an "Observe" option, specifying the /s resource when the default stream is selected.

Each response payload carries one or multiple notifications. The number of notification reported and the conditions used to remove notifications from the reported list is left to the implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero.

An example implementation is:

> Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After appending the instance, the content of the instance is sent to all clients observing the modified stream.
>
> Dependending on the storage space allocated to the notification stream, the oldest notifications that do not fit inside the notification stream storage space are removed.

```
FORMAT:
  Get /<stream-resource> Observe(0)

  2.05 Content (Content-Format :application/yang-tree+cbor)
  ordered map of instance-identifier, data-node-value
```

The array of data node instances may contain identical entries which have been generated at different times.

5.5.1. Notify Examples

Suppose the server generates the event specified in Appendix C.4. By executing a GET on the /s resource the client receives the following response:

```
REQ:  GET /s Observe(0) Token(0x93)

RES:  2.05 Content (Content-Format :application/yang-tree+cbor)
                        Observe(12) Token(0x93)
[
  60010,                  / example-port-fault (SID 60010) /
    {
      +1 : "0/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 2"   / port-fault (SID 60012) /
    },
  +0,                     / example-port-fault (SID 60010) /
    {
      +1 : "1/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 5"   / port-fault (SID 60012) /
    }
]
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send confirmable notifications periodically. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The request payload contains the values assigned to the input container when

specified. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1. Root data nodes are encoded using the delta between the current SID and the SID of the invoked instance identifier a specified by the URI.

The returned success response code is 2.05 Content.

```
FORMAT:
  POST /c/<instance identifier>
           (Content-Format :application/yang-value+cbor)
  data-node-value

  2.05 Content (Content-Format :application/yang-value+cbor)
  data-node-value
```

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix C.2. A server list is specified and the action "reset" (SID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
              (Content-Format :application/yang-value+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
}

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
{
  +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
}
```

6. Access to MIB Data

Appendix C.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```
{
   60021 :                    / list ipNetToPhysicalEntry /
   [
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType /
       60024 : h'0A000033',   / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,       / ipNetToPhysicalLastUpdated /
       60027 : 4,             / ipNetToPhysicalType /
       60028 : 1,             / ipNetToPhysicalState /
       60029 : 1              / ipNetToPhysicalRowStatus /
     },
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType /
       60024 : h'09020304',   / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,       / ipNetToPhysicalLastUpdated /
       60027 : 3,             / ipNetToPhysicalType /
       60028 : 6,             / ipNetToPhysicalState /
       60029 : 1              / ipNetToPhysicalRowStatus /
     }
   ]
}
```

In this example one instance of /ip/ipNetToPhysicalEntry (SID 60021, base64: Oz1) that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: CQIDBA) is requested.

```
REQ: GET example.com/c/Oz1?k="1,1,CQIDBA"

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
{
  +1 : 1,                  / ( SID 60022 ) /
  +2 : 1,                  / ( SID 60023 ) /
  +3 : h'09020304',        / ( SID 60024 ) /
  +4 : h'00000A36200A',    / ( SID 60025 ) /
  +5 : 2329836,            / ( SID 60026 ) /
  +6 : 3,                  / ( SID 60027 ) /
  +7 : 6,                  / ( SID 60028 ) /
  +8 : 1                   / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of data need to be transported, datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. On the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the resource, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with the actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length, which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.datastore" [RFC6690]. Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, the value "/c" is used as an example. The example below shows the discovery of the presence and location of management data.

REQ: GET /.well-known/core?rt=core.c.datastore

RES: 2.05 Content
</c>; rt="core.c.datastore"

Implemented data nodes MAY be discovered using the standard CoAP resource discovery. The implementation can add the data node identifiers (SID) supported to /.well-known/core with

rt="core.c.datanode". The available SIDs can be discovered by sending a GET request to "/.well-known/core" including a resource type (rt) parameter with the value "core.c.datanode". Upon success, the return payload will contain the registered SIDs and their location.

The example below shows the discovery of the presence and location of data nodes.

REQ: GET /.well-known/core?rt=core.c.datanode

RES: 2.05 Content
</c/BaAiN>; rt="core.c.datanode",
</c/CF_fA>; rt="core.c.datanode"

The list of data nodes may become prohibitively long. Therefore, it is recommended to discover the details about the YANG modules implemented by reading a YANG module library (e.g. "ietf-comi-yang-library" ad defined by [I-D.veillette-core-yang-library]).

The resource "/mod.uri" is used to retrieve the location of the YANG module library. This library can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

The following example shows the URI of a local instance of container modules-state (SID=1802) as defined in [I-D.veillette-core-yang-library].

REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example.com/c/cK

The following example shows the URI of a remote instance of same container.

REQ: GET example.com/mod.uri

RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
example-remote-server.com/group17/cK

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code.

Errors returned by a CoMI server can be broken into two categories, those associated to the CoAP protocol itself and those generated during the validation of the YANG data model constrains as described in [RFC7950] section 8.

The following list of common CoAP errors should be implemented by CoMI servers. This list is not exhaustive, other errors defined by CoAP and associated RFCs may be applicable.

- Error 4.01 (Unauthorized) is returned by the CoMI server when the CoMI client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).
- Error 4.02 (Bad Option) is returned by the CoMI server when one or more CoAP options are unknown or malformed.
- Error 4.04 (Not Found) is returned by the CoMI server when the CoMI client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).
- Error 4.05 (Method Not Allowed) is returned by the CoMI server when the CoMI client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).
- Error 4.08 (Request Entity Incomplete) is returned by the CoMI server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.
- Error 4.13 (Request Entity Too Large) may be returned by the CoMI server during a block transfer request, see [RFC7959] for more details.
- Error 4.15 (Unsupported Content-Format) is returned by the CoMI server when the Content-Format used in the request don’t match those specified in section 2.3.

CoMI server MUST also enforce the different constraints associated to the YANG data models implemented. These constraints are described in [RFC7950] section 8. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as

payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using delta value equal to the SID of the current schema node minus the SID of the parent container (i.e 1024).

```
+--rw error!
   +--rw error-tag             identityref
   +--rw error-app-tag?        identityref
   +--rw data-node-in-error?   instance-identifier
   +--rw error-message?        string
```

The following error-tag and error-app-tag are defined by the ietf-comi YANG module, these tags are implemented as YANG identity and can be extended as needed.

- error-tag operation-failed is returned by the CoMI server when the operation request cannot be processed successfully.
  - error-app-tag malformed-message is returned by the CoMI server when the payload received from the CoMI client don't contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or don't comply with the CBOR structure defined within this document.
  - error-app-tag data-not-unique is returned by the CoMI server when the validation of the 'unique' constraint of a list or leaf-list fails.
  - error-app-tag too-many-elements is returned by the CoMI server when the validation of the 'max-elements' constraint of a list or leaf-list fails.
  - error-app-tag too-few-elements is returned by the CoMI server when the validation of the 'min-elements' constraint of a list or leaf-list fails.
  - error-app-tag must-violation is returned by the CoMI server when the restrictions imposed by a 'must' statement are violated.
  - error-app-tag duplicate is returned by the CoMI server when a client tries to create a duplicate list or leaf-list entry.
- error-tag invalid-value is returned by the CoMI server when the CoMI client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.

  * error-app-tag invalid-datatype is returned by the CoMI server when CBOR encoding don't follow the rules set by or when the value is incompatible with the YANG Built-In type. (e.g. a value greater than 127 for an int8, undefined enumeration)

  * error-app-tag not-in-range is returned by the CoMI server when the validation of the 'range' property fails.

  * error-app-tag invalid-length is returned by the CoMI server when the validation of the 'length' property fails.

  * error-app-tag pattern-test-failed is returned by the CoMI server when the validation of the 'pattern' property fails.

- error-tag missing-element is returned by the CoMI server when the operation requested by a CoMI client fail to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.

  * error-app-tag missing-key is returned by the CoMI server to further qualify an missing-element error. This error is returned when the CoMI client tries to create or list instance, without all the 'key' specified or when the CoMI client tries to delete a leaf listed as a 'key'.

  * error-app-tag missing-input-parameter is returned by the CoMI server when the input parameters of an RPC or action are incomplete.

- error-tag unknown-element is returned by the CoMI server when the CoMI client tries to access a data node of a YANG module not supported, of a data node associated to an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

- error-tag bad-element is returned by the CoMI server when the CoMI client tries to create data nodes for more than one case in a choice.

- error-tag data-missing is returned by the CoMI server when a data node required to accept the request is not present.

  * error-app-tag instance-required is returned by the CoMI server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.

    * error-app-tag missing-choice is returned by the CoMI server when no nodes exist in a mandatory choice.

  o error-tag error is returned by the CoMI server when an unspecified error has occurred.

For example, the CoMI server might return the following error.

```
RES:  4.00 Bad Request (Content-Format :application/yang-value+cbor)
{
 +4 : 1020,        / error-tag = invalid-value /
 +2 : 1012,        / error-app-tag = not-in-range /
 +1 : 1736,        / data-node-in-error = timezone-utc-offset /
 +3 : "maximum value exceeded" / error-message /
}
```

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CoMI re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However, some adaptations may still be required, to cater for CoMI’s specific requirements.

11. IANA Considerations

11.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Value	Description	Reference
core.c.datastore	YANG datastore	RFC XXXX
core.c.datanode	YANG data node	RFC XXXX
core.c.liburi	YANG module library	RFC XXXX
core.c.eventstream	YANG event stream	RFC XXXX

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type	Excoding ID	Reference
application/yang-value+cbor	XXX	RFC XXXX
application/yang-values+cbor	XXX	RFC XXXX
application/yang-selectors+cbor	XXX	RFC XXXX
application/yang-tree+cbor	XXX	RFC XXXX
application/yang-ipatch+cbor	XXX	RFC XXXX

// RFC Ed.: replace XXX with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

Name	Template	Reference
yang-value+cbor	application/yang-value+cbor	RFC XXXX
yang-values+cbor	application/yang-values+cbor	RFC XXXX
yang-selectors+cbor	application/yang-selectors+cbor	RFC XXXX
yang-tree+cbor	application/yang-tree+cbor	RFC XXXX
yang-ipatch+cbor	application/yang-ipatch+cbor	RFC XXXX

Each of these media types share the following information:

- Subtype name: <as listed in table>
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See the Security Considerations section of RFC XXXX
- Interoperability considerations: N/A
- Published specification: RFC XXXX
- Applications that use this media type: CoMI
- Fragment identifier considerations: N/A
- Additional information:

  * Deprecated alias names for this type: N/A
  * Magic number(s): N/A
  * File extension(s): N/A
  * Macintosh file type code(s): N/A

- Person & email address to contact for further information: iesg&ietf.org

- o Intended usage: COMMON
- o Restrictions on usage: N/A
- o Author: Michel Veillette, ietf&augustcellars.com
- o Change Controller: IESG
- o Provisional registration? No

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

11.4. Concise Binary Object Representation (CBOR) Tags Registry

This document adds the following tags to the "Concise Binary Object Representation (CBOR) Tags" registry.

Tag	Data Item	Semantics	Reference
xxx	array	Oedered map	RFC XXXX

// RFC Ed.: replace xxx by the assigned Tag and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

12. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

Timothy Carey has provided the text for Appendix D.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. References

13.1. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Turner, R., Minaburo, A., and A. Somaraju, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-01 (work in progress), May 2017.

[I-D.ietf-core-yang-cbor]
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-04 (work in progress), February 2017.

[I-D.veillette-core-yang-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-yang-library-00 (work in progress), January 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <http://www.rfc-editor.org/info/rfc7950>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <http://www.rfc-editor.org/info/rfc8040>.

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <http://www.rfc-editor.org/info/rfc8132>.

13.2. Informative References

[I-D.ietf-core-interfaces] Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-09 (work in progress), March 2017.

[netconfcentral] YUMAworks, "NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <http://www.rfc-editor.org/info/rfc4293>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[XML] W3C, "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[yang-cbor] Veillette, M., "yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2017-07-01.yang"
module ietf-comi {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-comi";
  prefix comi;

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:alexander@ackl.io>

     Peter van der Stok
     <mailto:consultancy@vanderstok.org>

     Andy Bierman
     <mailto:andy@yumaworks.com>";

  description
    "This module contains the different definitions required
     by the CoMI protocol.";
```

```
     revision 2017-07-01 {
        description
         "Initial revision.";
       reference
         "draft-ietf-core-comi";
     }

     identity error-tag {
       description
         "Base identity for error-tag.";
     }

     identity operation-failed {
       base error-tag;
       description
         "Returned by the CoMI server when the operation request
          can't be processed successfully.";
     }

     identity invalid-value {
       base error-tag;
       description
         "Returned by the CoMI server when the CoMI client tries to
          update or create a leaf with a value encoded using an
          invalid CBOR datatype or if the 'range', 'length',
          'pattern' or 'require-instance' constrain is not
          fulfilled.";
     }

     identity missing-element {
       base error-tag;
       description
         "Returned by the CoMI server when the operation requested
          by a CoMI client fails to comply with the 'mandatory'
          constraint defined. The 'mandatory' constraint is
          enforced for leafs and choices, unless the node or any of
          its ancestors have a 'when' condition or 'if-feature'
          expression that evaluates to 'false'.";
     }

     identity unknown-element {
       base error-tag;
       description
         "Returned by the CoMI server when the CoMI client tries to
          access a data node of a YANG module not supported, of a
          data node associated with an 'if-feature' expression
          evaluated to 'false' or to a 'when' condition evaluated
          to 'false'.";
```

```
}

identity bad-element {
  base error-tag;
  description
    "Returned by the CoMI server when the CoMI client tries to
     create data nodes for more than one case in a choice.";
}

identity data-missing {
  base error-tag;
  description
    "Returned by the CoMI server when a data node required to
     accept the request is not present.";
}

identity error {
  base error-tag;
  description
    "Returned by the CoMI server when an unspecified error has
    occurred.";
}

identity error-app-tag {
  description
    "Base identity for error-app-tag.";
}


identity malformed-message {
  base error-app-tag;
  description
    "Returned by the CoMI server when the payload received
     from the CoMI client don’t contain a well-formed CBOR
     content as defined in [RFC7049] section 3.3 or don’t
     comply with the CBOR structure defined within this
     document.";
}

identity data-not-unique {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
     ’unique’ constraint of a list or leaf-list fails.";
}

identity too-many-elements {
  base error-app-tag;
```

```
    description
      "Returned by the CoMI server when the validation of the
       'max-elements' constraint of a list or leaf-list fails.";
  }

  identity too-few-elements {
    base error-app-tag;
    description
      "Returned by the CoMI server when the validation of the
       'min-elements' constraint of a list or leaf-list fails.";
  }

  identity must-violation {
    base error-app-tag;
    description
      "Returned by the CoMI server when the restrictions
       imposed by a 'must' statement are violated.";
  }

  identity duplicate {
    base error-app-tag;
    description
      "Returned by the CoMI server when a client tries to create
       a duplicate list or leaf-list entry.";
  }

  identity invalid-datatype {
    base error-app-tag;
    description
      "Returned by the CoMI server when CBOR encoding is
       incorect or when the value encoded is incompatible with
       the YANG Built-In type. (e.g. value greater than 127
       for an int8, undefined enumeration).";
  }

  identity not-in-range {
    base error-app-tag;
    description
      "Returned by the CoMI server when the validation of the
       'range' property fails.";
  }

  identity invalid-length {
    base error-app-tag;
    description
      "Returned by the CoMI server when the validation of the
       'length' property fails.";
  }
```

```
identity pattern-test-failed {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
     'pattern' property fails.";
}

identity missing-key {
  base error-app-tag;
  description
    "Returned by the CoMI server to further qualify a
     missing-element error. This error is returned when the
     CoMI client tries to create or list instance, without all
     the 'key' specified or when the CoMI client tries to
     delete a leaf listed as a 'key'.";
}

identity missing-input-parameter {
  base error-app-tag;
  description
    "Returned by the CoMI server when the input parameters
     of a RPC or action are incomplete.";
}

identity instance-required {
  base error-app-tag;
  description
    "Returned by the CoMI server when a leaf of type
     'instance-identifier' or 'leafref' marked with
     require-instance set to 'true' refers to an instance
     that does not exist.";
}

identity missing-choice {
  base error-app-tag;
  description
    "Returned by the CoMI server when no nodes exist in a
     mandatory choice.";
}

container error {
  presence "Error paylaod";

  description
    "Optional payload of a 4.00 Bad Request CoAP error.";

  leaf error-tag {
    type identityref {
```

```
          base error-tag;
        }
        mandatory true;
        description
          "The enumerated error-tag.";
      }

      leaf error-app-tag {
        type identityref {
          base error-app-tag;
        }
        description
          "The application-specific error-tag.";
      }

      leaf data-node-in-error {
        type instance-identifier;
        description
          "When the error reported is caused by a specific data node,
           this leaf identifies the data node in error.";
      }

      leaf error-message {
        type string;
        description
          "A message describing the error.";
      }
    }
  }
  <CODE ENDS>
```

Appendix B. ietf-comi .sid file

```
  {
    "assignment-ranges": [
      {
        "entry-point": 1000,
        "size": 100
      }
    ],
    "module-name": "ietf-comi",
    "module-revision": "2017-07-01",
    "items": [
      {
        "type": "Module",
        "label": "ietf-comi",
        "sid": 1000
      },
```

```
{
  "type": "identity",
  "label": "/error-app-tag",
  "sid": 1001
},
{
  "type": "identity",
  "label": "/error-app-tag/data-not-unique",
  "sid": 1002
},
{
  "type": "identity",
  "label": "/error-app-tag/duplicate",
  "sid": 1003
},
{
  "type": "identity",
  "label": "/error-app-tag/instance-required",
  "sid": 1004
},
{
  "type": "identity",
  "label": "/error-app-tag/invalid-datatype",
  "sid": 1005
},
{
  "type": "identity",
  "label": "/error-app-tag/invalid-length",
  "sid": 1006
},
{
  "type": "identity",
  "label": "/error-app-tag/malformed-message",
  "sid": 1007
},
{
  "type": "identity",
  "label": "/error-app-tag/missing-choice",
  "sid": 1008
},
{
  "type": "identity",
  "label": "/error-app-tag/missing-input-parameter",
  "sid": 1009
},
{
  "type": "identity",
  "label": "/error-app-tag/missing-key",
```

```
        "sid": 1010
      },
      {
        "type": "identity",
        "label": "/error-app-tag/must-violation",
        "sid": 1011
      },
      {
        "type": "identity",
        "label": "/error-app-tag/not-in-range",
        "sid": 1012
      },
      {
        "type": "identity",
        "label": "/error-app-tag/pattern-test-failed",
        "sid": 1013
      },
      {
        "type": "identity",
        "label": "/error-app-tag/too-few-elements",
        "sid": 1014
      },
      {
        "type": "identity",
        "label": "/error-app-tag/too-many-elements",
        "sid": 1015
      },
      {
        "type": "identity",
        "label": "/error-tag",
        "sid": 1016
      },
      {
        "type": "identity",
        "label": "/error-tag/bad-element",
        "sid": 1017
      },
      {
        "type": "identity",
        "label": "/error-tag/data-missing",
        "sid": 1018
      },
      {
        "type": "identity",
        "label": "/error-tag/error",
        "sid": 1019
      },
      {
```

```
        "type": "identity",
        "label": "/error-tag/invalid-value",
        "sid": 1020
      },
      {
        "type": "identity",
        "label": "/error-tag/missing-element",
        "sid": 1021
      },
      {
        "type": "identity",
        "label": "/error-tag/operation-failed",
        "sid": 1022
      },
      {
        "type": "identity",
        "label": "/error-tag/unknown-element",
        "sid": 1023
      },
      {
        "type": "node",
        "label": "/error",
        "sid": 1024
      },
      {
        "type": "node",
        "label": "/error/data-node-in-error",
        "sid": 1025
      },
      {
        "type": "node",
        "label": "/error/error-app-tag",
        "sid": 1026
      },
      {
        "type": "node",
        "label": "/error/error-message",
        "sid": 1027
      },
      {
        "type": "node",
        "label": "/error/error-tag",
        "sid": 1028
      }
    ]
  }
```

Appendix C. YANG example specifications

This appendix shows five YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign.

C.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```
module ietf-system {                    // SID 1700
  container system {                    // SID 1715
    container clock {                   // SID 1734
      choice timezone {
        case timezone-name {
          leaf timezone-name {          // SID 1735
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {    // SID 1736
            type int16 {
            }
          }
        }
      }
    }
    container ntp {                     // SID 1750
      leaf enabled {                    // SID 1751
        type boolean;
        default true;
      }
      list server {                     // SID 1752
        key name;
        leaf name {                     // SID 1755
          type string;
        }
        choice transport {
          case udp {
            container udp {             // SID 1757
              leaf address {            // SID 1758
                type inet:host;
              }
              leaf port {               // SID 1759
                type inet:port-number;
              }
            }
```

```
            }
          }
          leaf association-type {        // SID 1753
            type enumeration {
              enum server {
              }
              enum peer {
              }
              enum pool {
              }
            }
          }
          leaf iburst {                  // SID 1754
            type boolean;
          }
          leaf prefer {                  // SID 1756
            type boolean;
            default false;
          }
        }
      }
    container system-state {             // SID 1716
      container clock {                  // SID 1717
        leaf current-datetime {          // SID 1719
          type yang:date-and-time;
        }
        leaf boot-datetime {             // SID 1718
          type yang:date-and-time;
        }
      }
    }
  }
```

C.2. server list

Taken over from [RFC7950] section 7.15.3.

```
   module example-server-farm {
     yang-version 1.1;
     namespace "urn:example:server-farm";
     prefix "sfarm";

     import ietf-yang-types {
       prefix "yang";
     }

     list server {                          // SID 60000
       key name;
       leaf name {                          // SID 60001
         type string;
       }
       action reset {                       // SID 60002
         input {
           leaf reset-at {                  // SID 60003
             type yang:date-and-time;
             mandatory true;
           }
         }
         output {
           leaf reset-finished-at {         // SID 60004
             type yang:date-and-time;
             mandatory true;
           }
         }
       }
     }
   }
```

C.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
   module ietf-interfaces {               // SID 1500
     container interfaces {               // SID 1505
       list interface {                   // SID 1533
         key "name";
         leaf name {                      // SID 1537
           type string;
         }
         leaf description {               // SID 1534
           type string;
         }
         leaf type {                      // SID 1538
           type identityref {
             base interface-type;
           }
           mandatory true;
         }

         leaf enabled {                   // SID 1535
           type boolean;
           default "true";
         }

         leaf link-up-down-trap-enable {  // SID 1536
           if-feature if-mib;
           type enumeration {
             enum enabled {
               value 1;
             }
             enum disabled {
               value 2;
             }
           }
         }
       }
     }
   }
```

C.4. Example-port

Notification example defined within this document.

```
   module example-port {
           ...
           notification example-port-fault {   // SID 60010
             description
               "Event generated if a hardware fault on a
                line card port is detected";
             leaf port-name {                  // SID 60011
               type string;
               description "Port name";
             }
             leaf port-fault {                 // SID 60012
               type string;
               description "Error condition detected";
             }
           }
         }
```

C.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```
   module IP-MIB {
     import IF-MIB {
       prefix if-mib;
     }
     import INET-ADDRESS-MIB {
       prefix inet-address;
     }
     import SNMPv2-TC {
       prefix smiv2;
     }
     import ietf-inet-types {
       prefix inet;
     }
     import yang-smi {
       prefix smi;
     }
     import ietf-yang-types {
       prefix yang;
     }

     container ip {                              // SID 60020
       list ipNetToPhysicalEntry {               // SID 60021
         key "ipNetToPhysicalIfIndex
              ipNetToPhysicalNetAddressType
              ipNetToPhysicalNetAddress";
         leaf ipNetToPhysicalIfIndex {           // SID 60022
```

```
          type if-mib:InterfaceIndex;
        }
        leaf ipNetToPhysicalNetAddressType {  // SID 60023
          type inet-address:InetAddressType;
        }
        leaf ipNetToPhysicalNetAddress {      // SID 60024
          type inet-address:InetAddress;
        }
        leaf ipNetToPhysicalPhysAddress {     // SID 60025
          type yang:phys-address {
            length "0..65535";
          }
        }
        leaf ipNetToPhysicalLastUpdated {     // SID 60026
          type yang:timestamp;
        }
        leaf ipNetToPhysicalType {            // SID 60027
          type enumeration {
            enum "other" {
              value 1;
            }
            enum "invalid" {
              value 2;
            }
            enum "dynamic" {
              value 3;
            }
            enum "static" {
              value 4;
            }
            enum "local" {
              value 5;
            }
          }
        }
        leaf ipNetToPhysicalState {           // SID 60028
          type enumeration {
            enum "reachable" {
              value 1;
            }
            enum "stale" {
              value 2;
            }
            enum "delay" {
              value 3;
            }
            enum "probe" {
              value 4;
```

```
            }
            enum "invalid" {
              value 5;
            }
            enum "unknown" {
              value 6;
            }
            enum "incomplete" {
              value 7;
            }
          }
        }
        leaf ipNetToPhysicalRowStatus {       // SID 60029
          type smiv2:RowStatus;
      }  // list ipNetToPhysicalEntry
    }  // container ip
  }  // module IP-MIB
```

Appendix D. Comparison with LWM2M

TO DO Need updated text based on the current version of CoMI.
Multiple assumptions used in the original text are no more valid.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliantinc.com

Peter van der Stok (editor)
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com