CoRE   P. van der Stok
Internet-Draft   consultant
Intended status: Standards Track   A. Bierman
Expires: July 30, 2017   YumaWorks
                                                            M. Veillette
                                                  Trilliant Networks Inc.
                                                                A. Pelov
                                                                  Acklio
                                                        January 26, 2017

CoAP Management Interface
draft-ietf-core-comi-00

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 data 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

Table of Contents

1. Introduction

   The Constrained Application Protocol (CoAP) [RFC7252] is designed for
   Machine to Machine (M2M) applications such as smart energy 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. The 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 the draft [I-D.ietf-netconf-restconf] which
   describes a REST-like interface called RESTCONF, which uses HTTP
   methods to access structured data defined in YANG.

   The use of standardized data sets, specified in a standardized
   language such as YANG, promotes interoperability between devices and
   applications from different manufacturers. A large amount of
   Management Information Base (MIB) [mibreg] specifications already
   exists for monitoring purposes. This data can be accessed in
   RESTCONF or CoMI if the server converts the SMIv2 modules to YANG,
   using the mapping rules defined in [RFC6643].

   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 packets, 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].

Readers of this specification should be familiar with all the terms
and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]:
client, configuration data, datastore, and server.

The following terms are defined in the YANG data modelling language
[RFC7950]: anydata, anyxml, container, data node, key, key leaf,
leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol
[I-D.ietf-netconf-restconf]: data resource, datastore resource, edit
operation, query parameter, and target resource.

The following terms are defined in this document:

data node instance: An instance of a data node specified in a YANG
   module present in the server. The instance is stored in the
   memory of the server.

Notification instance: An instance of a schema node of type
   notification, specified in a YANG module present in the server.
   The instance is generated in the server at the occurrence of the
   corresponding event and appended to a stream.

YANG schema item identifier: Numeric identifier which replaces the
   name identifying a YANG item (see section 6.2 of [RFC7950])
   (anydata, anyxml, data node, RPC, Action, Notification, Identity,
   Module name, Submodule name, Feature).

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 submodule and 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: Value assigned to a data node instance. Data node
values are encoded based on the rules defined in section 4 of
[I-D.ietf-core-yang-cbor].

set of data node instances: Represents the payload of CoAP methods
when a collection is sent or returned. There are two
possibilities, dependent on Request context:

1. CBOR array of pair(s) <instance identifier, data node value >

2. CBOR map of pair(s) <instance identifier, data node value >

TODO: Reduce to one, if possible

The following list contains the abbreviations used in this document.

SID: YANG Schema Item iDentifier.

2. CoMI Architecture

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

```
+------------------------------------------------------------+
|               SMIv2  specification (2)                     |
+------------------------------------------------------------+
                           \ /
+------------------------------------------------------------+
|               YANG  specification  (1)                     |
+---------*-------------------------------------*----------+
          | compilation                         | compilation
          |               security (7)          |
client   \|/        [===========================]   \|/      Server
+-------------------+                      +-----------------+
|                   |                      |                 |
| Request generation +--> CoAP request(3) -->| Request retrieval|
| answer retrieval  |<-- CoAP response(3)<--| answer generation|
|       (4)         |                      |        (4)      |
+-------------------+                      | +---------------+|
                                           | | datastore (5) ||
                                           | +---------------+|
                                           |                 |
                                           |     Variable    |
                                           |Instrumentation(6)|
                                           +-----------------+
```
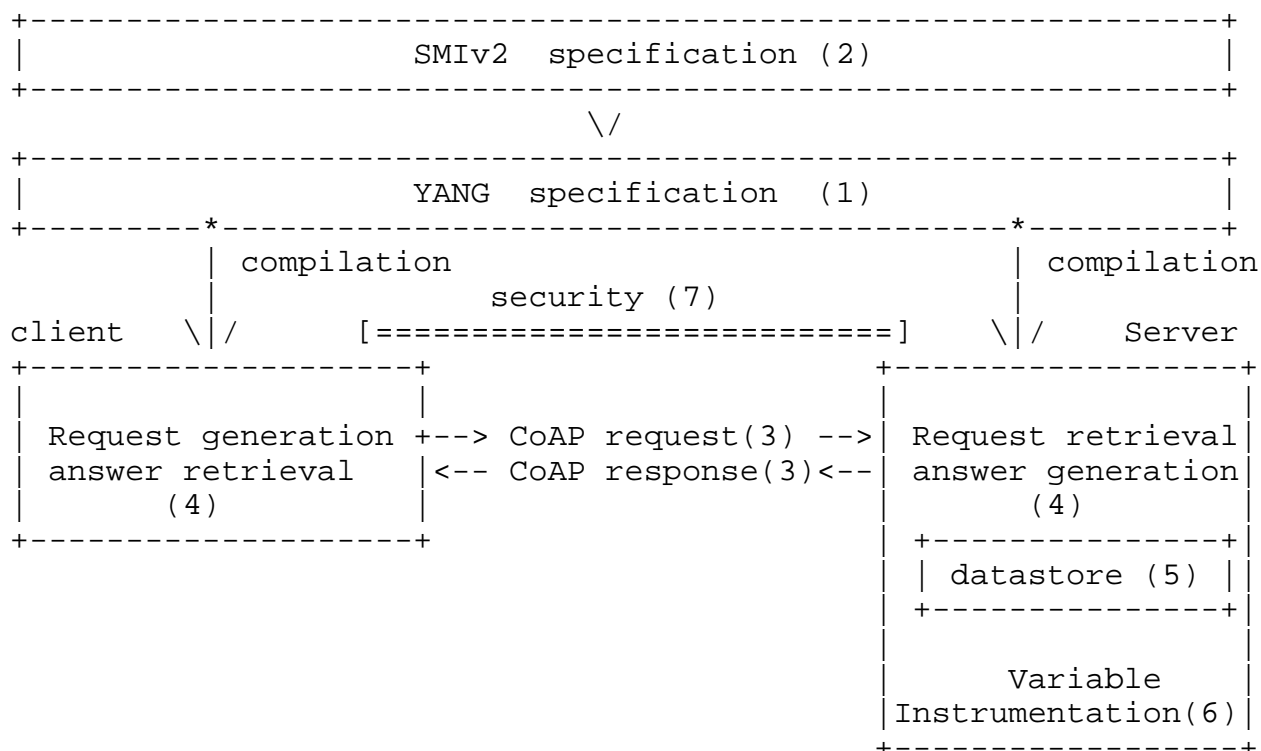
Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the
CoAP management architecture. A client sends requests as payload in
packets over the network to a managed constrained node.

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

(4) Retrieval, generation: The server and client parse the CoMI
    request/response and identify the corresponding instances in the
    datastore based on YANG specification.

(5) Datastore: The store is composed of two parts: Operational state and Configuration datastore. Datastore also supports RPCs and event streams.

(6) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.

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

2.1. Major differences between RESTCONF and CoMI

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 servers cannot change the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

CoMI servers only implement the efficient "trim" mode for default values.

The CoMI servers do not support the following RESTCONF functionality:

o The "fields" query parameter to query multiple instances.

o The 'filter' query that involves XML parsing, 'content', and 'depth', query parameters.

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 object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

Examples of object identifier encoding formats are described in [I-D.ietf-core-sid].

3. Example syntax

   This section presents the notation used for the examples. The YANG
   specifications that are used throughout this document are shown in
   Appendix A. The example specifications are taken over 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.

   A YANG (item identifier, item value) pair is mapped to a CBOR (key,
   value) pair. The YANG item value is encoded as specified in
   [I-D.ietf-core-yang-cbor]. The YANG item identifier can be a SID
   (single node identifier) or a CBOR array with the structure [SID,
   key1, key2] (list node identifier), where SID is a list identifier
   and the key values specify the list instance. The YANG item value
   can be any CBOR major type.

   Delta encoding is used for the SIDs. The notation +n is used when
   the SID has the value PREC+n where PREC is the SID of the parent
   container, or PREC is the SID of the preceding entity in a CBOR
   array.

   In all examples the resource path in the URI is expressed as a SID,
   represented as a base64 number. SIDs in the payload are represented
   as decimal numbers.

4. CoAP Interface

   In CoAP a group of links can constitute a Function Set.

   TODO: what will happen to term Function Set ?

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

   The path prefix /c has resources accessible with the following three
   paths:

   /c: YANG-based data with path "/c" and using CBOR content encoding
      format. This path represents a datastore resource which contains

YANG data resources as its descendant nodes. The data nodes are identified with their SID with format /c/SID.

/c/mod.uri: URI identifying the location of the server module information, with path "/c/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag 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.

/c/s: String identifying the default stream resource to which YANG notification instances are appended. 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: A YANG module describes a set of data trees composed of YANG data nodes. Every data node of the YANG modules loaded in the CoMI server represents a resource of the datastore container (e.g. /c/<sid>

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

TODO; reference to fetch and patch content formats.

The profile of the management function set, with IF=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/c	core.c	n/a
Data	/c	core.c.data	application/cbor
Module Set URI	/c/mod.uri	core.c.moduri	application/cbor
Events	/c/s	core.c.stream	application/cbor

5. /c Function Set

The /c Function Set provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

```
+----------+-------------------------------------------------------+
| Operation | Description                                          |
+----------+-------------------------------------------------------+
| GET      | Retrieve the datastore resource or a data resource    |
|          |                                                       |
| FETCH    | Retrieve (partial) data resource(s)                   |
|          |                                                       |
| POST     | Create a data resource, invoke RPC                    |
|          |                                                       |
| PUT      | Create or replace a data resource                     |
|          |                                                       |
| iPATCH   | Idem-potently create, replace, and delete data        |
|          | resource(s) (partially)                               |
|          |                                                       |
| DELETE   | Delete a data resource                                |
+----------+-------------------------------------------------------+
```

There is one query parameters for the GET, PUT, POST, and DELETE
methods.

```
+-----------------+------------------------------------+
| Query Parameter | Description                        |
+-----------------+------------------------------------+
| k               | Select an instance of a list node  |
+-----------------+------------------------------------+
```

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

5.1. Using the 'k' query parameter

The "k" (key) parameter specifies the instance of a list node. The
SID in the URI is followed by the (?k=key1, key2,..). Where SID
identifies a list node, and key1, key2 are the values of the key
leaves that specify an instance of the list.

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

YANG datatype	Binary representation	Text representation
uint8,uint16,unit32, uint64	CBOR unsigned integer	int_to_text(number)
int8, int16,int32, int64	CBOR negative integer	base64 (CBOR representation)
decimal64	CBOR decimal fractions	base64 (CBOR representation
string	CBOR text or string	text
boolean	CBOR false or true	"0" or "1"
enumeration	CBOR unsigned integer	int_to_text (number)
bits	CBOR byte string	base64 (CBOR representation)
binary	CBOR byte string	base64 (binary value)
identityref	CBOR unsigned integer	int_to_text (number)
union		base64 (CBOR representation)
List instance identifier	CBOR unsigned integer	base64 (CBOR representation)
List instance identifier	CBOR array	Base64 (CBOR representation)

5.2. Data Retrieval

   One or more data node instances 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
   [I-D.ietf-core-etch].

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] is
used, as explained in more detail in Section 7.

CoMI uses the FETCH payload for retrieving a subset of the datastore.

There are two additional query parameters for the GET and FETCH
methods.

```
+-----------+-------------------------------------------------------+
| Query     | Description                                           |
| Parameter |                                                       |
+-----------+-------------------------------------------------------+
| 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' query parameter

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 resources. A 4.00 Bad Request error is returned if used for
other methods or resource types.

If this query parameter is not present, the default value is "a".

5.2.2. Using the 'd' query parameter

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 yet, the server MUST return the leaf.

If the target of a GET method is a data node that represents a
container or list that has any child resources with default values,
for the child resources that have not been given value yet, the
server MUST not return the child resource if this query parameter is
set to 't' and MUST return the child resource if this query parameter
is set to 'a'.

If this query parameter 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. A single instance identifier is
specified in the URI path prefixed with /c.

FORMAT:
    GET /c/<instance identifier>

    2.05 Content (Content-Format: application/cbor)
    <data node value>

The returned payload is composed of all the children associated with
the specified data node instance.

The instance identifier is a SID or a SID followed by the "k" query
parameter.

5.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix A.1, a
request is sent to retrieve the value of system-state/clock/current-
datetime specified in container system-state. The ID of system-

state/clock/current-datetime is 1719, encoded in base64 this yields
a3. The answer to the request returns a <value>, transported as a
single CBOR string item.


```
REQ: GET example.com/c/a3

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

For example, the GET of the clock node (ID = 1717; base64: a1), sent
by the client, results in the following returned value sent by the
server, transported as a CBOR map containing 2 pairs:


```
REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/cbor)
{
      +2 : "2014-10-26T12:16:51Z",   / ID 1719 /
      +1 : "2014-10-21T03:00:00Z"    / ID 1718 /
}
```

A "list" node can have multiple instances. Accordingly, the returned
payload of GET is composed of all the instances associated with the
selected list node.

For example, look at the example in Appendix A.3. The GET of the
/interfaces/interface/ (with identifier 1533, base64: X9) results in
the following returned payload, transported as a CBOR array with 2
elements.

```
REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/cbor)
[
 {+4 : "eth0",                  / name  (ID 1537) /
  +1 : "Ethernet adaptor",     / description (ID 1534) /
  +5 : 1179,                    / type, (ID 1538) identity /
                                / ethernetCsmacd (ID 1179) /
  +2 : true                     / enabled ( ID 1535) /
 },
 {+4 : "eth1",                  / name (ID 1537) /
  +1 : "Ethernet adaptor",     / description (ID 1534) /
  +5 : 1179,                    / type, (ID 1538) identity /
                                / ethernetCsmacd (ID 1179) /
  +2 : false                    / enabled /
 }
]
```

It is equally possible to select a leaf of one instance of a list or
a complete instance container with GET. The instance identifier is
the numeric identifier of the list followed by the specification of
the values for the key leaves that uniquely identify the list
instance. The instance identifier looks like: SID?k=key-value. The
key of "interface" is the "name" leaf. The example below requests
the description leaf of the instance with name="eth0" (ID=1534,
base64: X-). The value of the description leaf is returned.


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

RES: 2.05 Content (Content-Format: application/cbor)
"Ethernet adaptor"
```

5.2.4. FETCH

The FETCH is used to retrieve a list of data node values. The FETCH
Request payload contains a CBOR list of instance identifiers.

```
FORMAT:
    FETCH /c/ Content-Format (application/YANG-fetch+cbor)
    <CBOR array of instance identifiers>

    2.05 Content (Content-Format: application/YANG-patch+cbor)
    <CBOR array of data node values>
```

The instance identifier is a SID or a CBOR array containing the SID
followed by key values that identify the list instance (sec 5.13.1 of
[I-D.ietf-core-yang-cbor]. In the payload of the returned data node

values, delta encoding is used as described in
[I-D.ietf-core-yang-cbor].

5.2.4.1. FETCH examples

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


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

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

TODO: align with future FETCH content format.

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 arrays so
messages will preserve their order.

5.3.2. POST

Data resources are created with the POST method. The CoAP POST
operation is used in CoMI for creation of data 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 the values of an instance of a container or leaf
is sent with a confirmable CoAP POST message. A single SID is
specified in the URI path prefixed with /c.

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

        2.01 Created (Content-Format: application/cbor)

If the data resource already exists, then the POST request MUST fail
and a "4.09 Conflict" status-line MUST be returned

The instance identifier is a SID or a SID followed by the "k" query
parameter.

5.3.2.1. Post example

The example uses the interface list from Appendix A.1. Example is
creating a new version of the container interface (ID = 1533):

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

RES: 2.01 Created (Content-Format: application/cbor)
```

5.3.3. PUT

Data resource instances are created or replaced with the PUT method.
The PUT operation is supported in CoMI. A request to set the value
of a data node instance is sent with a confirmable CoAP PUT message.

    FORMAT:
        PUT /c/<instance identifier> Content-Format(application/cbor)
        <data node value>

        2.01 Created

The instance identifier is a SID or a SID followed by the "k" query
parameter.

5.3.3.1. PUT example

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

   REQ: PUT /c/X9?k="eth0" Content-Format(application/cbor)
         {
           +4 : "eth0",   / name (ID 1537) /
           +1 : "Ethernet adaptor", / description (ID 1534) /
           +5 : 1179,   / type (ID 1538), identity /
                                     / ethernetCsmacd (ID 1179) /
           +2 : true   / enabled (ID 1535) /
         }
   RES: 2.04 Changed

5.3.4. iPATCH

   One or multiple data resource instances are replaced with the idem-
   potent iPATCH method [I-D.ietf-core-etch]. A request is sent with a
   confirmable CoAP iPATCH message.

   There are no query parameters for the iPATCH method.

   The processing of the iPATCH command is specified by the CBOR
   payload. The CBOR patch payload describes the changes to be made to
   target YANG data nodes [I-D.bormann-appsawg-cbor-merge-patch]. If
   the CBOR patch payload contains data node instances that are not
   present in the target, these instances are added or silently ignored
   dependent of the payload information. If the target contains the
   specified instance, the contents of the instances are replaced with
   the values of the payload. Null values indicate the removal of
   existing values.

   FORMAT:
       iPATCH /c Content-Format(application/YANG-patch+cbor)
       <set of data node instances>

       2.04 Changed

5.3.4.1. iPATCH example

   The example uses the interface list from Appendix A.3, and the
   timezone-utc-offset leaf from Appendix A.1. In the example one leaf
   (timezone-utc-offset) and one container (interface) instance are
   changed.

```
REQ: iPATCH /c Content-Format(application/YANG-patch+cbor)
[
   [1533, "eth0"] ,                     / interface (ID = 1533) /
     {
        +4 : "eth0",                    / name (ID 1537) /
        +1 : "Ethernet adaptor",        / description (ID 1534) /
        +5 : 1179,                      / type (ID 1538), identity /
                                        / ethernetCsmacd (ID 1179) /
        +2 : true                       / enabled (ID 1535) /
     },
   +203 , 60           / timezone-utc-offset (delta = 1736-1533) /
]
```

RES: 2.04 Changed

TODO: Align with future cbor-merge-patch content format.

5.3.5. DELETE

Data resource instances are deleted with the DELETE method. The
RESTCONF DELETE operation is supported in CoMI.

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

    2.02 Deleted
```

The instance identifier is a SID or a SID followed by the "k" query
parameter.

5.3.5.1. DELETE example

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

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

RES: 2.02 Deleted

5.4. Full Data Store access

The methods GET, PUT, POST, and DELETE can be used to return,
replace, create, and delete the whole data store respectively.

```
FORMAT:
   GET /c
   2.05 Content (Content-Format: application/cbor)
   <array of data node instances>

   PUT /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.04 Changed

   POST /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.01 Created

   DELETE /c
   2.02 Deleted
```

The array of data node instances represents an array of all root
nodes in the data store after the PUT, POST and GET method
invocations.

5.4.1. Full Data Store examples

The example uses the interface list and the clock container from
Appendix A.3. Assume that the data store contains two root objects:
the list interface (ID 1533) with one instance and the container
Clock (ID 1717). After invocation of GET an array with these two
objects is returned:

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

5.5. Notify functions

   Notification by the server to a selection of clients when an event
   occurs in the server is an essential function for the management of
   servers. CoMI allows events specified in YANG [RFC5277] to be
   notified to a selection of requesting clients. The server appends
   newly generated events to a stream. There is one, so-called
   "default", stream in a CoMI server. The /c/s resource identifies the
   default stream. The server MAY create additional stream resources.
   When a CoMI server generates an internal event, it is appended to the
   chosen stream, and the content of a notification instance is ready to
   be sent to all CoMI clients which observe the chosen stream resource.

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

   Every time an event is generated, the chosen stream is cleared, and
   the generated notification instance is appended to the chosen
   stream(s). After appending the instance, the contents of the
   instance is sent to all clients observing the modified stream.

```
   FORMAT:
     Get /<stream-resource>
         Content-Format(application/YANG-patch+cbor) Observe(0)

   2.05 Content Content-Format(application/YANG-patch+cbor)
   <set of data node instances>
```

5.5.1. Notify Examples

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

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

   RES: 2.05 Content Content-Format(application/YANG-patch+cbor)
                         Observe(12) Token(0x93)
   {
      60010 :   / example-port-fault (ID 60010) /
        {
          +1 : "0/4/21",   / port-name (ID 60011) /
          +2 : "Open pin 2"   / port-fault (ID 60012) /
        },
      60010 :   / example-port-fault (ID 60010) /
        {
          +1 : "1/4/21",   / port-name (ID 60011) /
          +2 : "Open pin 5"   / port-fault (ID 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 once in a while. 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 with the action station. The Response payload contains the
   values of the output container when specified.

   The returned success response code is 2.05 Content.

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

   2.05 Content Content-Format (application/YANG-patch+cbor)
   <output node value>
```

There "k" query parameter is allowed for the POST method when used for an action invocation.

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix A.2. A server list is specified and the action "reset" (ID 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-patch+cbor)
   {
     +1 : "2016-02-08T14:10:08Z09:00" / reset-at (ID 60003) /
   }

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

6. Access to MIB Data

Appendix A.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: ipv4 /
         60024 : h'0A000033',         / ipNetToPhysicalNetAddress /
         60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
         60026 : 2333943,             / ipNetToPhysicalLastUpdated /
         60027 : 4,                   / ipNetToPhysicalType: static /
         60028 : 1,                   / ipNetToPhysicalState: reachable /
         60029 : 1                    / ipNetToPhysicalRowStatus: active /
       },
       {
         60022 : 1,                   / ipNetToPhysicalIfIndex /
         60023 : 1,                   / ipNetToPhysicalNetAddressType: ipv4 /
         60024 : h'09020304',         / ipNetToPhysicalNetAddress  /
         60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
         60026 : 2329836,             / ipNetToPhysicalLastUpdated /
         60027 : 3,                   / ipNetToPhysicalType: dynamic /
         60028 : 6,                   / ipNetToPhysicalState: unknown /
         60029 : 1                    / ipNetToPhysicalRowStatus: active /
       }
     ]
  }
```

The IPv4 addresses A.0.0.33 and 9.2.3.4 are encoded in CBOR as
h'0A000033' and h'09020304' respectively. In the following example
exactly one instance is requested from the ipNetToPhysicalEntry (ID
60021, base64: Oz1). The h'09020304' value is encoded in base64 as
AJAgME.

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

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

RES: 2.05 Content (Content-Format: application/YANG-patch+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 text need to be transported
   the 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. In the server,
   values are changed, lists are re-ordered, extended or reduced. When
   these actions happen during the serialization of the contents of the
   variables, 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
   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" [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, but it is recommended that the value "/c" is used,
where possible. The example below shows the discovery of the
presence and location of management data.

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

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

Management objects MAY be discovered with the standard CoAP resource
discovery. The implementation can add the encoded values of the
object identifiers to /.well-known/core with rt="core.c.data". The
available objects identified by the encoded values can be discovered
by sending a GET request to "/.well-known/core" including a resource
type (RT) parameter with the value "core.c.data". Upon success, the
return payload will contain the registered encoded values and their
location. The example below shows the discovery of the presence and
location of management data.

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

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

Lists of encoded values may become prohibitively long. It is
discouraged to provide long lists of objects on discovery.
Therefore, it is recommended that details about management objects
are discovered by reading the YANG module information stored in for
example the "ietf-comi-yang-library" module
[I-D.veillette-core-cool-library]. The resource "/c/mod.uri" is used
to retrieve the location of the YANG module library.

The module list 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.

      Local in example.com server:

      REQ: GET example.com/c/mod.uri

      RES: 2.05 Content (Content-Format: application/cbor)
      {
        "mod.uri" : "example.com/c/modules"
      }

      Remote in example-remote-server:

      REQ: GET example.com/c/mod.uri

      RES: 2.05 Content (Content-Format: application/cbor)
      {
        "moduri" : "example-remote-server.com/c/group17/modules"
      }

   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, and SHOULD include
   additional information in the payload.

   Such an error message payload is a text string, using the following
   structure:

   CoMI error: xxxx "error text"

   The characters xxxx represent one of the values from the table below,
   and the OPTIONAL "error text" field contains a human readable
   explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.xx	General error
1	4.13	Request too big
2	4.00	Response too big
3	4.00	Unknown identifier
4	4.00	Invalid value
5	4.05	Attempt to write read-only variable
6	5.01	No access
7	4.00	Wrong type
8	4.15	Unknown encoding
9	4.0	Wrong value
10	4.0	Not created
11	4.04	Resource unavailable
12	4.01	Authorization error
13	4.0	Bad attribute
14	4.0	Unknown attribute
15	4.0	Missing attribute

The CoMI error codes are motivated by the error-status values defined in [RFC3416], and the error tags defined in [I-D.ietf-netconf-restconf].

10. Security Considerations

   For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. 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

'rt="core.c"' needs registration with IANA.

'rt="core.c.data"' needs registration with IANA.

'rt="core.c.moduri"' needs registration with IANA.

'rt="core.c.stream"' needs registration with IANA.

Content types to be registered:

o application/YANG-patch+cbor

o application/YANG-fetch+cbor

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 B.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Juergen

Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. Changelog

Copy of vanderstok-core-comi-11.

14. References

14.1. Normative References

[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>.

[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>.

[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>.

[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>.

[I-D.ietf-netconf-restconf]
          Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
          Protocol", draft-ietf-netconf-restconf-18 (work in
          progress), October 2016.

[I-D.ietf-core-etch]
          Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch
          Methods for Constrained Application Protocol (CoAP)",
          draft-ietf-core-etch-04 (work in progress), November 2016.

[I-D.bormann-appsawg-cbor-merge-patch]
          Bormann, C. and P. Stok, "CBOR Merge Patch", draft-
          bormann-appsawg-cbor-merge-patch-00 (work in progress),
          March 2016.

[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-03 (work in progress), October
          2016.

14.2. Informative References

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J.
          Schoenwaelder, Ed., "Structure of Management Information
          Version 2 (SMIv2)", STD 58, RFC 2578,
          DOI 10.17487/RFC2578, April 1999,
          <http://www.rfc-editor.org/info/rfc2578>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart,
          "Introduction and Applicability Statements for Internet-
          Standard Management Framework", RFC 3410,
          DOI 10.17487/RFC3410, December 2002,
          <http://www.rfc-editor.org/info/rfc3410>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations
          for the Simple Network Management Protocol (SNMP)",
          STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002,
          <http://www.rfc-editor.org/info/rfc3416>.

[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>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed.,
          and A. Bierman, Ed., "Network Configuration Protocol
          (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011,
          <http://www.rfc-editor.org/info/rfc6241>.

   [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>.

   [RFC6643]   Schoenwaelder, J., "Translation of Structure of Management
               Information Version 2 (SMIv2) MIB Modules to YANG
               Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012,
               <http://www.rfc-editor.org/info/rfc6643>.

   [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>.

   [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-07 (work in
               progress), December 2016.

   [I-D.ietf-core-sid]
               Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A.
               Minaburo, "YANG Schema Item iDentifier (SID)", draft-ietf-
               core-sid-00 (work in progress), October 2016.

   [I-D.veillette-core-cool]
               Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A.
               Minaburo, "Constrained Objects Language", draft-veillette-
               core-cool-02 (work in progress), July 2016.

   [I-D.veillette-core-cool-library]
               Veillette, M., "Constrained YANG Module Library", draft-
               veillette-core-cool-library-00 (work in progress), August
               2016.

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

   [OMA]   "OMA-TS-LightweightM2M-V1_0-20131210-C", Web
                 http://technical.openmobilealliance.org/Technical/
                 current_releases.aspx.

   [OMNA]   "Open Mobile Naming Authority (OMNA)", Web
                 http://http://technical.openmobilealliance.org/Technical/
                 technical-information/omna.

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

   [mibreg]   "Structure of Management Information (SMI) Numbers (MIB
                 Module Registrations)", Web
                 http://www.iana.org/assignments/smi-numbers/
                 smi-numbers.xhtml/.

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

Appendix A. YANG example specifications

   This appendix shows 5 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, taken from [yang-cbor].

A.1. ietf-system

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

```
   module ietf-system {
     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;
          }
        }
      }
    }


A.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;
           }
         }
       }
     }
   }


A.3.  interfaces

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

```
module ietf-interfaces {
  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;
          }
        }
      }
    }
  }
}
```

A.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";
            }
          }
        }
```

A.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 ipNetToPhysi
calNetAddress";
      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 B. Comparison with LWM2M

B.1. Introduction

   CoMI and LWM2M [OMA], both, provide RESTful device management
   services over CoAP. Differences between the designs are highlighted
   in this section.

   The intent of the LWM2M protocol is to provide a single protocol to
   control and manage IoT devices. This means the IoT device implements
   and uses the same LWM2M agent function for the actuation and sensing
   features of the IoT device as well as for the management of the IoT
   device. The intent of CoMI Interface as described in the Abstract
   section of this document is to provide management of constrained
   devices and devices in constrained networks using RESTCONF and YANG.
   This implies that the device, although reusing the CoAP protocol,
   would need a separate CoAP based agent in the future to control the
   actuation and sensing features of the device and another CoMI agent
   that performs the management functions.

   It should be noted that the mapping of a LWM2M server to YANG is
   specified in [YANGlwm2m]. The converted server can be invoked with
   CoMI as specified in this document.

   For the purposes of managing IoT devices the following points related
   to the protocols compare how management resources are defined,
   identified, encoded and updated.

B.2. Defining Management Resources

   Management resources in LWM2M (LWM2M objects) are defined using a
   standardized number. When a new management resource is defined,
   either by a standards organization or a private enterprise, the
   management resource is registered with the Open Mobile Naming
   Authority [OMNA] in order to ensure different resource definitions do
   not use the same identifier. CoMI, by virtue of using YANG as its
   data modeling language, allows enterprises and standards
   organizations to define new management resources (YANG nodes) within
   YANG modules without having to register each individual management
   resource. Instead YANG modules are scoped within a registered name
   space. As such, the CoMI approach provides additional flexibility in
   defining management resources. Likewise, since CoMI utilizes YANG,
   existing YANG modules can be reused. The flexibility and reuse
   capabilities afforded to CoMI can be useful in management of devices
   like routers and switches in constrained networks. However for
   management of IoT devices, the usefulness of this flexibility and
   applicability of reuse of existing YANG modules may not be warranted.
   The reason is that IoT devices typically do not require complex sets
   of configuration or monitoring operations required by devices like a
   router or a switch. To date, OMA has defined approximately 15
   management resources for constrained and non-constrained mobile or
   fixed IoT devices while other 3rd Party SDOs have defined another 10
   management resources for their use in non-constrained IoT devices.
   Likewise, the Constrained Object Language [I-D.veillette-core-cool]
   which is used by CoMI when managing constrained IoT devices uses YANG
   schema item identifiers, which are registered with IANA, in order to
   define management resources that are encoded using CBOR when
   targeting constrained IoT Devices.

B.3. Identifying Management Resources

   As LWM2M and CoMI can similarly be used to manage IoT devices,
   comparison of the CoAP URIs used to identify resources is relevant as
   the size of the resource URI becomes applicable for IoT devices in
   constrained networks. LWM2M uses a flat identifier structure to
   identify management resources and are identified using the LWM2M
   object's identifier, instance identifier and optionally resource
   identifier (for access to and object's attributes). For example,
   identifier of a device object (object id = 3) would be "/3/0" and
   identification of the device object's manufacturer attribute would be
   "/3/0/0". Effectively LWM2M identifiers for management resources are
   between 4 and 10 bytes in length.

   CoMI is expected to be used to manage constrained IoT devices. CoMI
   utilizes the YANG schema item identifier[SID] that identify the
   resources. CoMI recommends that IoT device expose resources to

identify the data stores and event streams of the CoMI agent.
Individual resources (e.g., device object) are not directly
identified but are encoded within the payload. As such the
identifier of the CoMI resource is smaller (4 to 7 bytes) but the
overall payload size isn't smaller as resource identifiers are
encoded on the payload.

B.4. Encoding of Management Resources

LWM2M provides a separation of the definition of the management
resources from how the payloads are encoded. As of the writing of
this document LWM2M encodes LWM2M encodes payload data in Type-
length-value (TLV), JSON or plain text formats. JSON encoding is the
most common encoding scheme with TLV encoding used on the simplest
IoT devices. CoMI's use of CBOR provides a more efficient transfer
mechanism [RFC7049] than the current LWM2M encoding formats.

In situations where resources need to be modified, CoMI uses the CoAP
PATCH operation resources only require a partial update. LWM2M does
not currently use the CoAP PATCH operation but instead uses the CoAP
PUT and POST operations which are less efficient.

Authors' Addresses

   Peter van der Stok
   consultant

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

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

   Email: andy@yumaworks.com

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

Phone: +14503750556
Email: michel.veillette@trilliantinc.com

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

Email: a@ackl.io