core
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

P. van der Stok
consultant
B. Greevenbosch
Huawei Technologies
A. Bierman
YumaWorks
J. Schoenwaelder
A. Sehgal
Jacobs University
October 27, 2014

CoAP Management Interface
draft-vanderstok-core-comi-05

Abstract

This document describes a network management interface for constrained devices, called CoMI. CoMI is an adaptation of the RESTCONF protocol for use in constrained devices and networks. It is designed to reduce the message sizes, server code size, and application development complexity. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. The payload of the CoMI message is encoded in Concise Binary Object Representation (CBOR).

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 April 30, 2015.

Copyright Notice

Copyright (c) 2014 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 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.

The draft [I-D.ietf-netconf-restconf] describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG [RFC6020]. RESTCONF allows access to data resources contained in NETCONF [RFC6241] datastores. RESTCONF messages can be encoded in XML [XML] or JSON. The GET method is used to retrieve data resources and the POST, PUT, PATCH, and DELETE methods are used to create, replace, merge, and delete data resources.

A large amount of Management Information Base (MIB) [RFC3418] specifications already exist for monitoring purposes. This data can be accessed in RESTCONF if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion. The

RESTCONF protocol is adapted and optimized for use in constrained environments, using CoAP instead of HTTP. Standardized data sets promote interoperability between small devices and applications from different manufacturers. Stateless communication is encouraged to keep communications simple and the amount of state information small in line with the design objectives of 6lowpan [RFC4944] [RFC6775], RPL [RFC6650], and CoAP [RFC7252].

RESTCONF uses the HTTP methods HEAD, OPTIONS, and PATCH, which are not available in CoAP. The use of TCP is also a problem in HTTP. The transport protocols available fo CoAP are much better suited to constrained networks.

CoMI is low resource oriented, uses CoAP, and only supports the methods GET, PUT, POST and DELETE. CoMI uses CBOR as the data format. To support small payloads, CoMI use an additional data identifier string to number conversion to minimise CBOR payloads. It is assumed that the managed device is the most constrained entity. The client might be more capable, however this is not necessarily the case.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP. When the MIB can be accessed with the CoAP protocol, the SNMP protocol can be replaced with the CoAP protocol. Although the SNMP server size is not huge (see Appendix A), the code for the security aspects of SMIv3 is not negligible. Using CoAP to access secured management objects reduces the code complexity of the stack in the constrained device, and harmonizes applications development.

The objective of CoMI is to provide a CoAP based Function Set that reads and sets values of managed objects in devices to (1) initialize parameter values at start-up, (2) acquire statistics during operation, and (3) maintain nodes by adjusting parameter values during operation.

The payload of CoMI is encoded in CBOR [RFC7049] which is automatically generated from JSON [JSON]. CBOR has a binary format and hence has more coding efficiency than JSON.

The end goal of CoMI is to provide information exchange over the CoAP transport protocol in a uniform manner as a first step to the full

management functionality as specified in [I-D.ersue-constrained-mgmt].

1.1. Design considerations

CoMI supports discovery of resources, accompanied by reading, writing and notification of resource values. CoMI supports MIB modules which have been translated from SMIv2 to YANG, using [RFC6643]. This mapping is read-only so writable SMIv2 objects need to be converted to YANG using an implementation-specific mapping.

CoMI uses a simple URI to access the management object resources. Complexity introduced by module name, context specification, or row selection, is expressed with uri-query attributes. The choice for uri-query attributes makes the URI structure less context dependent.

1.2. 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

server

The following terms are defined in the YANG data modeling language [RFC6020]:

container

data node

key

key leaf

leaf

leaf-list

list

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]:

data resource

datastore resource

edit operation

query parameter

target resource

unified datastore

The following terms are defined in this document:

YANG hash: CoMI object identifier, which is a 32-bit numeric hash of the YANG object identifier string for the object. When a YANG hash value is printed in a request target URI, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

1.2.1. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

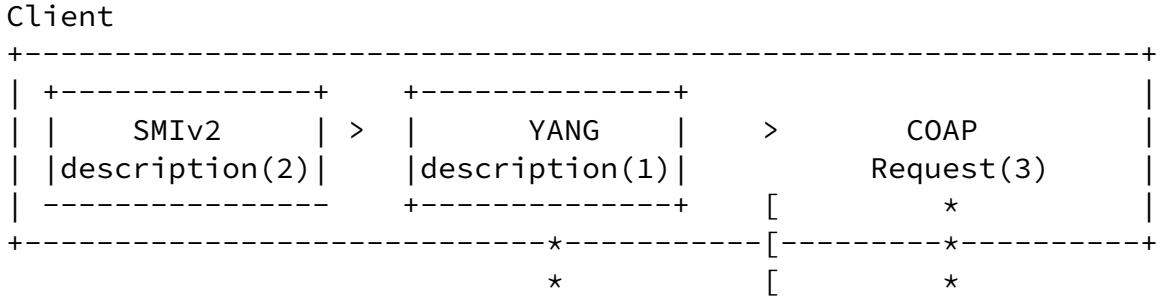
Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying of instrumentation variables used for the management of the instrumented node.



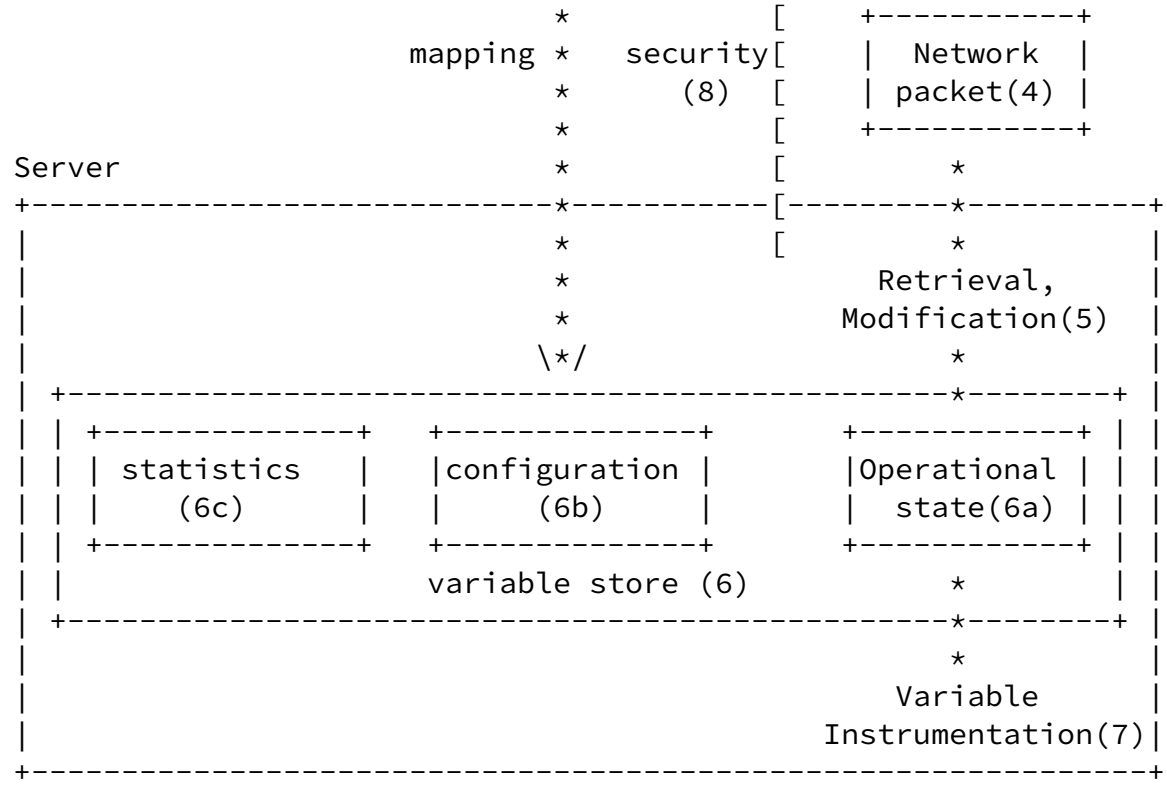


Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

Objectives are:

- Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.

- The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.

- The client receives the performance characteristics on a regular basis or on request.

- The client sets the parameter values in the server at bootstrap

and intermittently when operational conditions change.

- The constrained network requires the payload to be as small as possible, and the constrained server memory requirements should be as small as possible.

The components in Figure 1 have the following high-level functionality.

- The instrumentation variables and parameters are described in the YANG or SMIv2 language.

- Descriptions in the SMIv2 language are translated into the YANG language.

- The YANG description serves as input to the writers of application and instrumentation code and the humans analysing the returned values (arrow from YANG description to Variable store). The description is used to check the correctness of the CoAP request and do the CBOR encoding.

- The CoAP request packs the request to read or set variables in the packet payload, which is transmitted over the network using IP. The CoAP request also receives values returned by the server in the payload of a packet. On arrival of the packet in the server, the payload is retrieved and decoded.

- Values are stored in the appropriate variables in the Variable store, and or values are returned from the Variable store into the payload of the packet. The Variable instrumentation code stores the values of the parameters into the appropriate places in the operational code. The variable instrumentation code reads current execution values from the operational code and stores them in the Variable store.

- The network communication must be secured.

For interoperability it is required that in addition to using the Internet Protocol for data transport:

- The names, type, and semantics of the instrumentation variables are standardized.

- The instrumentation variables are described in a standard language.

- The signature of the CoAP request in the server is standardized.

- The format of the packet payload is standardized.
- The notification from server to client is standardized.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG description: A description contains a set of named and versioned modules. A module contains a hierarchy of named and typed resources. A resource is uniquely identified by a sequence of its name and the names of the enveloping resources following the hierarchy order.

(2) SMIv2 description: A named module contains a set of variables and "conceptual tables". Named variables have simple types. Conceptual tables are composed of typed named columns. The variable name and module name identify the variable uniquely. There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request: The CoAP request needs a Universal Resource Identifier (URI) and the payload of the packet to send a request. The URI is composed of the schema, server, path and query and looks like coap://entry.example.com/<path>?<query>. Fragments are not supported. Allowed operations are PUT, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option can be used to return variable values regularly or on event occurrence (notification).

(3.1) CoAP <path>: The path identifies the variable in the form "/mg/data/<identifier>.

(3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the module name, the smi context, and others. The idea is to keep the path simple and put variations on variable specification in the query.

(3.3) CoAP discovery: Discovery of the variables is done with standard CoAP resource discovery using /.well-known/core with ?rt=/core/mg.

(4) Network packet: The payload contains the CBOR encoding of a single JSON object. This object corresponds to the converted RESTCONF message payload.

(5) Retrieval, modification: The server needs to parse the CBOR

encoded message and identify the corresponding entries in the

Variable store. In addition, this component includes the code for CoAP Observe and block options.

(6) Variable store: The store is composed of three parts: Operational state, Configuration datastore, and Statistics (see Section 2.1.

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

(8) Security: The server MUST prevent unauthorized users from reading or writing any data resources, however a standard access control model for CoMI should be specified in a separate document. DTLS is specified to secure CoAP communication.

2.1. RESTCONF/YANG Architecture

CoMI adapts the RESTCONF architecture so data exchange and implementation requirements are optimized for constrained devices.

The RESTCONF protocol uses a unified datastore to edit conceptual data structures supported by the server. The details of transaction preparation and non-volatile storage of the data are hidden from the RESTCONF client. CoMI also uses a unified datastore, to allow stateless editing of any configuration variables.

The child schema nodes of the unified datastore include all the top-level YANG data nodes in all the YANG modules supported by the server. The YANG data structures represent a hierarchy of data resources. The client discovers the list of YANG modules, and important conformance information such as the module revision dates, YANG features supported, and YANG deviations required. The individual data nodes are discovered indirectly by parsing the YANG modules supported by the server.

The YANG data definition statements contain a lot of information that can help automation tools, developers, and operators use the data model correctly and efficiently. The YANG definitions and server YANG module capability advertisements provide an "API contract" that

allow a client to determine the detailed server management capabilities very quickly. CoMI allows access to the same data resources as a RESTCONF server, except the messages are optimized to reduce identifier and payload size.

RESTCONF uses a simple algorithmic mapping from YANG to URI syntax to identify the target resource of a retrieval or edit operation. A client can construct operations or scripts using a predictable syntax, based on the YANG data definitions. The target resource URI

can reference a data resource instance, or the datastore itself (to retrieve the entire datastore or create a top-level data resource instance). CoMI uses a 32-bit YANG hash value (based on the YANG data node path identifier strings) to identify schema nodes in the target resource URI.

Any message payload data is relative to the node specified in the target resource URI in a request message. Each message is a well-formed XML instance document or JSON object, corresponding to the YANG schema definition specified by the target resource node. CoMI uses the hash value for the data node identifier in the message payloads, instead of the module-name prefixed identifier name like RESTCONF. CoMI message payloads are based on the JSON encoding of a RESTCONF message payload. The JSON identifier names are first converted to their 32-bit YANG hash values and then the payload is converted to CBOR.

3. CoAP Interface

In CoRE a group of links can constitute a 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.mg, with path: /mg, where mg is short-hand for management.

The mg resource has three sub-resources accessible with the paths:

/mg/data: YANG-based data with path "/mg/data" and using CBOR content encoding format. This path represents a datastore resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this

path are encoded as YANG hash values.

/mg/moduri: URI indicating the location of the server module information, with path "/mg/moduri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG hash values.

/mg/yang-hash: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has path "/mg/yang-hash" and is encoded in CBOR format. The "ietf-yang-hash" module in Section 5.3 is used to define the syntax and semantics of this data structure. This YANG data is encoded with plain identifier strings, not YANG hash values. The server will only have this resource if there are any objects that needed to be re-hashed due to a hash collision.

The "/mg/data" resource provides access to the YANG data resources as described in Section 4. The "/mg/moduri" resource provides access to a URI that can be used to retrieve an instance of the "ietf-yang-library" module, defined in the RESTCONF draft. This module lists the name, revision, and conformance information for each YANG module, which a client needs to determine the YANG data model objects that are supported by a server. This URI can be reference a local data resource within the server, or reference a remote data resource, such as a shared file server. The data resource identifiers are YANG hash values, as described in Section 5.1.

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

name	path	rt	Data Type
Management	/mg	core.mg	n/a
Data	/mg/data	core.mg.data	application/cbor
Module Set URI	/mg/moduri	core.mg.moduri	application/cbor

YANG Hash Info	/mg/yang-hash	core.mg.yang-hash	application/cbor

4. MG Function Set

The MG Function Set provides a CoAP interface to perform a subset of the functions provided by RESTCONF.

A subset of the operations defined in RESTCONF are used in CoMI:

Operation	Description
GET	Retrieve the datastore resource or a data resource
POST	Create a data resource
PUT	Create or replace a data resource
DELETE	Delete a data resource

4.1. Data Retrieval

4.1.1. GET

Data resources are retrieved by the client with the GET method. The RESTCONF GET operation is supported in CoMI. The same constraints apply as defined in section 3.3 of [I-D.ietf-netconf-restconf]. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252].

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 [I-D.ietf-core-block] is used. 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.

There are optional query parameters for the GET method. A CoMI server MAY implement these query parameters in order to allow common data retrieval filtering functionality.

Query Parameter	Description
content	Select config and/or non-config data resources
depth	Request limited sub-tree depth in the reply content
select	Request selected sub-trees from the target resource

4.1.2. Mapping of the 'select' Parameter

TODO: discuss how only a limited subset of the select parameter is used, and how the node names are changed to YANG hashes.

4.1.3. Retrieval Examples

The examples in this section use a JSON payload with one or more entries describing the pair (objectID, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 5.

4.1.3.1. Single object values

A request to read the value of a management object or the leaf of an object is sent with a confirmable CoAP GET message. A single object is specified in the URI path prefixed with /mg/data.

A request to set the value of an object/leaf is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request.

Using for example the clock container from [RFC7317], a request is

sent to retrieve the value of clock/current-datetime specified in module system-state. The answer to the request returns a (ObjectID, value) pair.

In all examples: (1) the payload is expressed in JSON, although the operational payload is specified to be in CBOR, and (2) the path is expressed in readable names although the transported path is expressed in numbers.

```
REQ: GET example.com/mg/data/system-state/clock/current-datetime

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

TODO: convert the example above so it uses YANG hash values instead of the string "current-datetime". Convert the target resource URI string "system-state/clock/current-datetime" to a YANG hash value.

The specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. Each leaf is returned as a (YANG hash, value) pair. For example, the GET of the clock object, sent by the client, results in the following returned payload sent by the managed entity:


```
REQ: GET example.com/mg/data/system-state/clock
   (Content-Format: application/cbor)

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

```
{
   "clock" : {
      "current-datetime" : "2014-10-26T12:16:51Z",
      "boot-datetime" : "2014-10-21T03:00:00Z"
      "timezone" : {
         "timezone-location" : "Europe/Stockholm",
         "timezone-utc-offset" : -60
      }
   }
}
```

TODO: convert the example above so it uses YANG hash values instead of the strings "clock", "current-datetime", "boot-datetime", "timezone", and "timezone-utc-offset". Convert the target resource URI string "system-state/clock" to a YANG hash value.

The specified object can be a list. Accordingly, the returned payload is composed of all the entries associated with the list. Each entry is returned as a (entry name, value) pair. For example, the GET of the ietf-ip/ipv6/neighbor list, sent by the client, results in the following returned payload sent by the managed entity:

```
REQ: GET example.com/mg/data/ietf-ip/ipv6/neighbor
   (Content-Format: application/cbor)


RES: 2.05 Content (Content-Format: application/cbor)
{
   "neighbor" : [
     {
    "ip" : "fe80::200:f8ff:fe21:67cf",
    "link-layer-address" : "00:00::10:01:23:45"
    },
     {
    "ip" : "fe80::200:f8ff:fe21:6708",
    "link-layer-address" : "00:00::10:54:32:10"
    },
     {
    "ip" : "fe80::200:f8ff:fe21:88ee",
    "link-layer-address" : "00:00::10:98:76:54"
    }
    ]
}
```

TODO: convert the example above so it uses YANG hash values instead of the strings "neighbor", "ip", and "link-layer-address". Convert the target resource URI string "ietf-ip/ipv6/neighbor" to a YANG hash value.

TODO: show examples using the "keys" parameter to select a specific instance of a list entry.

4.2. Data Editing

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

TODO: Should this be an optional feature? A server can choose to only support YANG modules with read-only objects. MIN-ACCESS conformance does not exist in YANG.

4.2.1. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI, however it is only allowed for creation of data resources. The same constraints apply as defined in section 3.4.1 of [I-D.ietf-netconf-restconf]. The operation is mapped to the POST method defined in section 5.8.2 of [RFC7252].

There are no query parameters for the POST method.

TODO: how to support user-ordered lists in YANG? This is done with the 'insert' and 'point' parameters in RESTCONF. In CoMI, a client will have to replace the all list or leaf-list entries (e.g. PUT parent container) to change the order or insert anywhere but last.

4.2.2. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PUT method defined in section 5.8.3 of [RFC7252].

There are no query parameters for the PUT method.

TODO: how to support user-ordered lists in YANG? Same issue as for POST.

4.2.3. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI. The same constraints apply as defined in section 3.7 of [I-D.ietf-netconf-restconf]. The operation is mapped to the DELETE method defined in section 5.8.4 of [RFC7252].

There are no optional query parameters for the PUT method.

4.3. Module Discovery

Management objects are discovered in a manner similar to the RESTCONF protocol, not with the standard CoAP resource discovery. Only the YANG module information needs to be retrieved by the client. The YANG modules contain all the data resource schema and naming information.

The "rt" attribute is used to filter resource queries as specified in [RFC6690].

The resource "/mg/moduri" is used to retrieve the location of the YANG module library information for the server. This data structure is defined in the "ietf-yang-library" module in the RESTCONF draft.

Since many constrained servers within a deployment are likely to be similar, the module list can be stored locally on each server, or remotely on a different server.

TODO: Example:

Local:

REQ: GET example.com/mg/moduri

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "moduri" : "example.com/mg/data/modules"
}
```

Remote:

REQ: GET example.com/mg/moduri

RES: 2.05 Content (Content-Format: application/cbor)

```
{
  "moduri" : "example-remote-server.com/mg/data/group17/modules"
}
```

4.4. Error Return Codes

The RESTCONF return status codes defined in section 6 of the RESTCONF draft are used in CoMI error responses, except they are converted to CoAP error codes.

TODO: complete RESTCONF to CoAP error code mappings

RESTCONF Status Line	CoAP Status Code
100 Continue	none?
200 OK	2.05
201 Created	2.01
202 Accepted	none?
204 No Content	?
304 Not Modified	2.03
400 Bad Request	4.00
403 Forbidden	4.03
404 Not Found	4.04
405 Method Not Allowed	4.05
409 Conflict	none?
412 Precondition Failed	4.12

413 Request Entity Too Large	4.13
414 Request-URI Too Large	4.00
415 Unsupported Media Type	4.15
500 Internal Server Error	5.00
501 Not Implemented	5.01
503 Service Unavailable	5.03

5. Mapping YANG to CoMI payload

A mapping for the encoding of YANG data in CBOR is necessary for the efficient transport of management data in the CoAP payload. Since object names may be rather long and may occur repeatedly, CoMI allows for association of a given object path identifier string value with an integer, called a "YANG hash".

5.1. YANG Hash Generation

The association between string value and string number is done through a hash algorithm. The "murmur3" 32-bit hash algorithm is used. This hash algorithm is described online at http://en.wikipedia.org/wiki/MurmurHash. Implementation are available online, including at https://code.google.com/p/smhasher/wiki/MurmurHash.

The hash is generated for the string representing the object path identifier. A canonical representation of the path identifier is used.

Prefix values are used on every node.

The prefix values defined in the YANG module containing the data object are used for the path expression. For external modules, this is the value of the 'prefix' sub-statement in the 'import' statement for each external module.

Path expressions for objects which augment data nodes in external modules are calculated in the augmenting module, using the prefix values in the augmenting module.

Choice and case node names are not included in the path expression. Only 'container', 'list', 'leaf', 'leaf-list', and 'anyxml' nodes are listed in the path expression.

The "murmur3_32" hash function is executed for the entire path string. The value '42' is used as the seed for the hash function.

The resulting 32-bit number is used by the server, unless the value is already being used for a different object by the server. In this case, the re-hash procedure in the following section is executed.

5.2. Re-Hash Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 77,000 objects in its YANG modules, then the probability of a collision is fairly high. If a hash collision occurs on the server, then the object that is causing the conflict has to be altered, such that the new hash value does not conflict with any value already in use by the server.

In most cases, the hash function is expected to produce unique values for all the objects supported by a constrained device. Given a known set of YANG modules, both server and client can calculate the YANG hashes independently, and offline.

Even though collisions are expected to happen rather rarely, they needs to be considered. Collisions can be detected before deployment, if the vendor knows which modules are supported by the server, and hence all YANG hashes can be calculated. Collisions are only an issue when they occur at the same server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any object identifiers, then it MUST create a "rehash-map" entry for the altered identifier, as described in the following YANG module.

5.3. ietf-yang-hash YANG Module

The "ietf-yang-hash" YANG module is used by the server to report any objects that have been mapped to produce a new hash value that does not conflict with any other YANG hash values used by the server.

YANG tree diagram for "ietf-yang-hash" module:

```
   +--ro yang-hash
      +--ro rehash* [hash]
         +--ro hash      uint32
         +--ro path?     string
         +--ro append?   string
```

```
module ietf-yang-hash {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-hash";
  prefix "yh";

  organization
    "IETF CORE (Constrained RESTful Environments) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/core/>
     WG List:  <mailto:core@ietf.org>

     WG Chair: Carsten Bormann
               <mailto:cabo@tzi.org>

     WG Chair: Andrew McGregor
               <mailto:andrewmcgr@google.com>

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



```
     Editor:   Bert Greevenbosch
               <mailto:andy@bert.greevenbosch@huawei.com>

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>
```

```
      Editor:   Juergen Schoenwaelder
                <mailto:j.schoenwaelder@jacobs-university.de>

      Editor:   Anuj Sehgal
                <mailto:s.anuj@jacobs-university.de>";

    description
      "This module contains re-hash information for the CoMI protocol.

       Copyright (c) 2014 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX; see
       the RFC itself for full legal notices.";

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

    // RFC Ed.: remove this note
    // Note: extracted from draft-vanderstok-core-comi-05.txt

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.
    revision 2014-10-27 {
      description
        "Initial revision.";
      reference
        "RFC XXXX: CoMI Protocol.";
    }

    container yang-hash {
      config false;
      description
        "Contains information on the YANG Hash values used by
         the server.";
```

```
      list rehash {
        key hash;
        description
          "Each entry describes an re-hash mapping in use by
           the server.";

        leaf hash {
          type uint32;
          description "The hash value that has a collision";
        }
        leaf path {
          type string;
          description
            "The YANG identifier path expression that caused the
             collision and is being remapped";
        }
        leaf append {
          type string;
          description
            "The string that the server appended to the path
             expression contained in the 'path' leaf to produce
             a new path expression and therefore new hash value.
             The YANG hash value for the new string (identified
             by 'path' + 'append') is used to identify the
             'path' object.";
        }
      }
    }

  }
```

5.3.1. YANG Re-Hash Example

In this example the server has an object that is already registered when the "/foo:A/foo:B/foo:col1" object is processed. This object path string hashes to value "a9abdcca". The server has appended the string "_" to the path to produce a new hash ("ea7a2044") which does not collide with any other objects.

The server would return the following information if the client retrieved the "/mg/yang-hash" resource.

```
REQ: GET example.com/mg/yang-hash

RES: 2.05 Content (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
      "rehash" : [
         {
            "hash" : 3933872196,
            "path" :"/foo:A/foo:B/foo:col1",
            "append" : "_"
         }
      ]
   }
}
```

5.4. The 'keys' Query Parameter

There is a mandatory query parameter that MUST be supported by servers called "keys". This parameter is used to specify the key values for an instance of an object identified by a YANG hash value. Any key leaf values for the object are passed in order. The first key leaf in the top-most list is the first key encoded in the 'keys' parameter.

Example: In this example the following YANG module is used:

```
  module foo-mod {
    namespace foo-mod-ns;
    prefix foo;

    list A {
      key "key1 key2";
      leaf key1 { type string; }
      leaf key2 { type int32; }
      list B {
        key "key3";
        leaf key3 { type string; }
        leaf col1 { type uint32; }
      }
    }
```

```
  }
```

The path identifier for the leaf "col1" is the following string:


```
/foo:A/foo:B/foo:col1
```

The YANG has value for this identifier string "2846612682" (hex "a9abdcca").

The following string represents the RESTCONF target resource URI expression for the "col1" leaf for the key values "top", 17, and "group1":

```
/restconf/data/foo-mod:A=top,17/B=group1/col1
```

The following string represents the CoMI target resource identifier for the same instance of the "col1" leaf:

```
/mg/data/a9abdcca?keys=top,17,group1
```

6. Mapping YANG to CBOR

6.1. Conversion from YANG datatypes to CBOR datatypes

Table 1 defines the mapping between YANG datatypes and CBOR datatypes.

Elements of types not in this table, and of which the type cannot be inferred from a type in this table, are ignored in the CBOR encoding by default. Examples include the "description" and "key" elements. However, conversion rules for some elements to CBOR MAY be defined elsewhere.

```
+--------------+------------------+---------------------------------+
```

YANG type	CBOR type	Specification
int8, int16, int32, int64, uint16, uint32, uint64, decimal64	unsigned int (major type 0) or negative int (mayor type 1)	The CBOR integer type depends on the sign of the actual value.
boolean	either "true" (major type 7, simple value 21) or "false" (major type 7, simple value 20)	
string	text string (major type 3)	
enumeration	unsigned int (major type 0)	
bits	array of text strings	Each text string contains the name of a bit value that is set.
binary	byte string (major type 2)	
empty	null (major type 7, simple value 22)	TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...
union		Similar ot the JSON transcription from [I-D.ietf-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].

leaf-list	array (major type 4)	The array is encapsulated in the map associated with the descriptor.
list	map (major type 5)	Like the higher level map, the lower level map contains descriptor number - value pairs of the elements in the list.
container	map (major type 5)	The map contains descriptor number - value pairs corresponding to the elements in the container.
smiv2:oid	array of integers	Each integer contains an element of the OID, the first integer in the array corresponds to the most left element in the OID.

Table 1: Conversion of YANG datatypes to CBOR

7. Examples

TODO: add examples back with YANG hash values and a YANG module.

8. Trap functions

A trap can be set through the CoAP Observe [I-D.ietf-core-observe] function. As regular with Observe, the client subscribes to the variable by sending a GET request with an "Observe" option.

TODO: Observe example

In the registration request, the client MAY include a "Response-To-Uri-Host" and optionally "Response-To-Uri-Port" option as defined in [I-D.becker-core-coap-sms-gprs]. In this case, the observations SHOULD be sent to the address and port indicated in these options. This can be useful when the client wants the managed device to send

the trap information to a multicast address.

9. Object access management

Two topics are relevant: (1) the definition of the destination of Notify messages, and (2) the creation and maintenance of "string to number" tables.

9.1. Notify destinations

The destination of notifications need to be communicated to the applications sending them. Draft [I-D.ietf-core-interfaces] describes the binding of end-points to end-points on remote devices. The object with type "binding table" contains a sequence of bindings. The contents of bindings contains the methods, location, the interval specifications, and the step value as suggested in [I-D.ietf-core-interfaces]. The method "notify" has been added to the binding methods "poll", "obs" and "push", to cater for the binding of notification source to the receiver.

TODO: describe interface for NOTIFY destination definition.

10. Error Handling

In case a request is received which cannot be processed properly, the managed entity 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 encoded in CBOR, using the following structure:

TODO: Adapt RESTCONF <errors> data structure for use in CoMI. Need to select the most important fields like <error-path>.

```
errorMsg     : ErrorMsg;

*ErrorMsg {
  errorCode  : uint;
```

```
  ?errorText : tstr;
}
```

The variable "errorCode" has one of the values from the table below, and the OPTIONAL "errorText" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.00	General error
1	4.00	Malformed CBOR data
2	4.00	Incorrect CBOR datatype
3	4.00	Unknown MIB variable
4	4.00	Unknown conversion table
5	4.05	Attempt to write read-only variable
0..2	5.01	Access exceptions
0..18	5.00	SMI error status

The CoAP error code 5.01 is associated with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

11. Security Considerations

For secure network management, it is important to restrict access to MIB variables only to authorised 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 for protected access to resources, as well suitable authentication and authorisation 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 authorisation 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.

12. IANA Considerations

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

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

'rt="core.mg.yang-hash"' needs registration with IANA.

Content types to be registered:

o application/comi+cbor

13. Acknowledgements

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR. The draft has benefited from comments (alphabetical order) by Dee Denteneer, Esko Dijk, Michael van Hartskamp, Zach Shelby, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka's description on conversion from YANG to JSON.

14. Changelog

Changes from version 00 to version 01

- Focus on MIB only
- Introduced CBOR, JSON, removed BER
- defined mappings from SMI to xx
- Introduced the concept of addressable table rows

Changes from version 01 to version 02

- Focus on CBOR, used JSON for examples, removed XML and EXI
- added uri-query attributes mod and con to specify modules and contexts
- Definition of CBOR string conversion tables for data reduction
- use of Block for multiple fragments
- Error returns generalized
- SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- Added security considerations

Changes from version 03 to version 04

- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05

- Merged SNMP access with RESTCONF access to management objects in small devices
- Added CoMI architecture section

- Added RESTCONf NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[I-D.becker-core-coap-sms-gprs] Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.

[I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-15 (work in progress), July 2014.

[I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.

[I-D.ietf-json-rfc4627bis] Bray, T., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-10 (work in progress), December 2013.

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-01 (work in progress), October 2014.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-02 (work in progress), October 2014.

15.2. Informative References

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", STD 17, RFC 1213, March 1991.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the

Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4088] Black, D., McCloghrie, K., and J. Schoenwaelder, "Uniform Resource Identifier (URI) Scheme for the Simple Network Management Protocol (SNMP)", RFC 4088, June 2005.

[RFC4113] Fenner, B. and J. Flick, "Management Information Base for the User Datagram Protocol (UDP)", RFC 4113, June 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4293] Routhier, S., "Management Information Base for the Internet Protocol (IP)", RFC 4293, April 2006.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, July 2012.

[RFC6650] Falk, J. and M. Kucherawy, "Creation and Use of Email Feedback Reports: An Applicability Statement for the Abuse Reporting Format (ARF)", RFC 6650, June 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link

Format", RFC 6690, August 2012.

[RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

[I-D.ietf-core-groupcomm] Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-25 (work in progress), September 2014.

[I-D.ietf-core-interfaces] Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-01 (work in progress), December 2013.

[I-D.ersue-constrained-mgmt] Ersue, M., Romascanu, D., and J. Schoenwaelder, "Management of Networks with Constrained Devices: Problem Statement, Use Cases and Requirements", draft-ersue-constrained-mgmt-03 (work in progress), February 2013.

[I-D.ietf-6lo-lowpan-mib] Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", draft-ietf-6lo-lowpan-mib-04 (work in progress), September 2014.

[I-D.ietf-lwig-coap] Kovatsch, M., Bergmann, O., Dijk, E., He, X., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-01 (work in progress), July 2014.

[STD0001] "Official Internet Protocols Standard", Web http://www.rfc-editor.org/rfcxx00.html, .

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

[JSON] "JavaScript Object Notation (JSON)", Web http://www.json.org, .

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

[DTLS-size] Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", Web http://www.vs.inf.ethz.ch/publ/papers/mshafagh_secon14.pdf, .

[dcaf] Bormann, C., Bergmann, O., and S. Gerdes, "Delegated Authenticated Authorization for Constrained Environments", Private Information , .

[openwsn] Watteijne, T., "Coap size in Openwsn", Web http://builder.openwsn.org/, .

[Erbium] Kovatsch, M., "Erbium Memory footprint for coap-18", Private Communication , .

[management] Schoenwalder, J. and A. Sehgal, "Management of the Internet of Things", Web http://cnds.eecs.jacobs-university.de/slides/2013-im-iot-management.pdf, 2013.

Appendix A. Payload and Server sizes

This section provides information on code sizes and payload sizes for a set of management servers. Approximate code sizes are:

Code	processor	Text	Data	reference
Observe agent	erbium	800	n/a	[Erbium]

CoAP server	MSP430	1K	6	[openwsn]
SNMP server	ATmega128	9K	700	[management]
Secure SNMP	ATmega128	30K	1.5K	[management]
DTLS server	ATmega128	37K	2K	[management]
NETCONF	ATmega128	23K	627	[management]
JSON parser	CC2538	4.6K	8	[dcaf]
CBOR parser	CC2538	1.5K	2.6K	[dcaf]
DTLS server	ARM7	15K	4	[I-D.ietf-lwig-coap]
DTLS server	MSP430	15K	4	[DTLS-size]
Certificate	MSP430	23K		[DTLS-size]
Crypto	MSP430	2-8K		[DTLS-size]

Thomas says that the size of the CoAP server is rather arbitrary, as its size depends mostly on the implementation of the underlying library modules and interfaces.

Payload sizes are compared for the following request payloads, where each attribute value is null (N.B. these sizes are educated guesses, will be replaced with generated data). The identifier are assumed to be a string representation of the OID. Sizes for SysUpTime differ due to preambles of payload. "CBOR opt" stands for CBOR payload where the strings are replaced by table numbers.

Request	BERR SNMP	JSON	CBOR	CBOR opt
IPnetTOMediaTable	205	327	~327	~51
lowpanIfStatsTable		710	614	121
sysUpTime	29	13	~13	20

RESTconf example				

Appendix B. Notational Convention for CBOR data

To express CBOR structures [RFC7049], this document uses the following conventions:

A declaration of a CBOR variable has the form:

```
name : datatype;
```

where "name" is the name of the variable, and "datatype" its CBOR datatype.

The name of the variable has no encoding in the CBOR data.

"datatype" can be a CBOR primitive such as:

tstr: A text string (major type 3)

uint: An unsigned integer (major type 0)

map(x,y): A map (major type 5), where each first element of a pair is of datatype x, and each second element of datatype y. A '.' character for either x or y means that all datatypes for that element are valid.

A datatype can also be a CBOR structure, in which case the variable's "datatype" field contains the name of the CBOR structure. Such CBOR structure is defined by a character sequence consisting of first its name, then a '{' character, then its subfields and finally a '}' character.

A CBOR structure can be encapsulated in an array, in which case its name in its definition is preceded by a '*' character. Otherwise the structure is just a grouping of fields, but without actual encoding of such grouping.

The name of an optional field is preceded by a '?' character. This

means, that the field may be omitted if not required.

Authors' Addresses

Peter van der Stok
consultant

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

Bert Greevenbosch
Huawei Technologies Co., Ltd.
Huawei Industrial Base
Bantian, Longgang District
Shenzhen 518129
P.R. China

Email: bert.greevenbosch@huawei.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: s.anuj@jacobs-university.de