Abstract

The draft describes an interface based on CoAP to manage constrained devices via MIBs. The proposed integration of CoAP with SNMP reduces the code- and application development complexity by accessing MIBs via a standard CoAP server. The payload of the MIB request is CBOR based on JSON. The mapping from SMI to CBOR is specified.

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 July 13, 2014.

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.
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 .................................................. 3
   1.1. Terminology .................................................. 4
2. CoAP Interface .................................................. 5
3. MIB Function Set .............................................. 5
   3.1. SNMP/MIB architecture ....................................... 5
       3.1.1. SNMP functions ......................................... 6
       3.1.2. MIB structure .......................................... 7
   3.2. CoMI Function Set ........................................... 8
       3.2.1. Single MIB values ....................................... 9
       3.2.2. multi MIB values ....................................... 11
       3.2.3. Table row .............................................. 13
       3.2.4. Error returns ......................................... 14
4. Mapping SMI to CoMI payload ................................. 14
   4.1. Mapping strings to CBOR ................................... 15
   4.2. Mapping SMI to CBOR ....................................... 16
       4.2.1. General overview ..................................... 16
       4.2.2. Conversion from YANG datatypes to CBOR datatypes .. 16
       4.2.3. Examples .............................................. 18
       4.2.4. 6LoWPAN MIB .......................................... 20
5. MIB discovery .................................................. 23
6. Trap functions .................................................. 24
7. MIB access management ......................................... 24
   7.1. Notify destinations ......................................... 24
   7.2. Conversion tables ......................................... 25
8. Error handling .................................................. 25
9. Security Considerations ........................................ 26
10. IANA Considerations ............................................ 26
11. Acknowledgements .............................................. 26
12. Changelog ...................................................... 27
13. References ..................................................... 27
   13.1. Normative References .................................... 27
   13.2. Informative References .................................. 28
Appendix A. Notational Convention for CBOR data ............... 30
Authors’ Addresses ................................................. 31
1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at Machine to Machine (M2M) applications such as smart energy and building control.

Small M2M devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected to be installed in future installations. The management protocol of choice for Internet is SNMP [RFC3410] as is testified by the large number of Management Information Base (MIB) [RFC3418] specifications currently published [STD0001]. More recently, the NETCONF protocol [RFC6241] was developed with an extended set of messages using XML [XML] as data format. The data syntax is specified with YANG [RFC6020] and a mapping from Yang to XML is specified. In [RFC6643] SMIv2 syntax is expressed in Yang. Contrary to SNMP and also CoAP, NETCONF assumes persistent connections for example provided by SSH. The NETCONF protocol provides operations to retrieve, configure, copy, and delete configuration data-stores. Configuring data-stores distinguishes NETCONF from SNMP which operates on standardized MIBs.

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion and is thus closer to SNMP than to NETCONF. 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 [I-D.ietf-core-coap].

The draft [I-D.bierman-netconf-restconf] describes a restful interface to NETCONF data stores and approaches the CoMI approach. CoMI uses SMI encoded in CBOR, and CoAP/UDP to access MIBs, where restconf uses YANG encoded in JSON and HTTP/TCP to access NETCONF data stores. CoMI is more low resource oriented than restconf is.

Currently, managed devices need to support 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. This arrangement 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 MIB variables 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 similar to JSON [JSON], but has a binary format and hence has more coding efficiency. CoMI is intended for small devices. The JSON overhead can be prohibitive. It is therefore chosen to transport CBOR in the payload. CBOR, like BER used for SNMP, transports the data type in the payload.

The end goal of CoMI is to provide information exchange over the CoAP transport protocol in a uniform manner to approach the full management functionality as specified in [I-D.ersue-constrained-mgmt].

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 are required to be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

Core Management Interface (CoMI) specifies the profile of Function Sets which access MIBs with the purpose of managing the operation of constrained devices in a network.

The following list defines the terms used in this document:

Managing Entity: An entity that manages one or more managed devices. Within the CoMI framework, the managing entity acts as a CoAP client for CoMI.

Managed Device: An entity that is being managed. The managed device acts as a CoAP server for CoMI.

NOTE: It is assumed that the managed device is the most constrained entity. The managing entity might be more capable, however this is not necessarily the case.

The following list contains the abbreviations used in this document.

OID: ASN.1 OBJECT-IDENTIFIER, which is used to uniquely identify MIB objects in the managed device.
2. 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 two sub-resources accessible with the paths:

- MIB with path /mg/mib and a CBOR content format.
- XLAT with path /mg/xlat and CBOR content format.

The mib resource provides access to the MIBs as described in Section 3.2. The xlat resource provides access to a string to CBOR identifier table as described in Section 4.1. The mib and xlat resources are introduced as sub resources to mg to permit later additions to CoMI mg resource.

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

<table>
<thead>
<tr>
<th>name</th>
<th>path</th>
<th>RT</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>/mg</td>
<td>core.mg</td>
<td>n/a</td>
</tr>
<tr>
<td>MIB</td>
<td>/mg/mib</td>
<td>core.mg.mib</td>
<td>application/cbor</td>
</tr>
<tr>
<td>XLAT</td>
<td>/mg/xlat</td>
<td>core.mg.xlat</td>
<td>application/cbor</td>
</tr>
</tbody>
</table>

3. MIB Function Set

The MIB Function Set provides a CoAP interface to perform equivalent functions to the ones provided by SNMP. Section 3.1 explains the structure of SNMP Protocol Data Units (PDU), their transport, and the structure of the MIB modules. An excellent overview of the documents describing the SNMP/MIB architecture is provided in section 7 of [RFC3410].

3.1. SNMP/MIB architecture

The architecture of the Internet Standard management framework consists of:
o A data definition language that is referred to as Structure of Management Information (SMI) [RFC2578].

o The Management Information Base (MIB) which contains the information to be managed and is defined for each specific function to be managed [RFC3418].

o A protocol definition referred to as Simple Network Management Protocol (SNMP) [RFC3416].

o Security and administration that provides SNMP message based security on the basis of the user-based security model [RFC3414].

o A management domain definition where a SNMP entity has access to a collection of management information called a "context" [RFC3411].

In addition [RFC4088] describes a URI scheme to refer to a specific MIB instance.

Separation in modules was motivated by the wish to respond to the evolution of Internet. The protocol part (SNMP) and data definition part (MIB) are independent of each other. The separation has enabled the progressive passage from SNMPv1 via SNMPv2 to SNMPv3. This draft leverages this separation to replace the SNMP protocol with a CoAP based protocol.

3.1.1. SNMP functions

The SNMP protocol supports seven types of access supported by as many Protocol Data Unit (PDU) types:

o Get Request, transmits a list of OBJECT-IDENTIFIERs to be paired with values.

o GetNext Request, transmits a list of OBJECT-IDENTIFIERs to which lexicographic successors are returned for table traversal.

o GetBulk Request, transmits a list of OBJECT-IDENTIFIERs and the maximum number of expected paired values.

o Response, returns an error or the (OBJECT-IDENTIFIER, value) pairs for the OBJECT-IDENTIFIERs specified in Get, GetNext, GetBulk, Set, or Inform Requests.

o Set Request, transmits a list of (OBJECT-IDENTIFIERs, value) pairs to be set in the specified MIB object.
3.1.2. MIB structure

A MIB module is composed of MIB objects. MIB objects are standardized by the IETF or by other relevant Standards Developing Organizations (SDO).

MIB objects have a descriptor and an identifier: OBJECT-IDENTIFIER (OID). The identifier, following the OSI hierarchy, is an ordered list of non-negative numbers [RFC2578]. OID values are unique. Each number in the list is referred as a sub-identifier. The descriptor is unique within a module. Different modules may contain the same descriptor. Consequently, a descriptor can be related to several OIDs.

Many instances of an object type exist within a management domain. Each instance can be identified within some scope or "context", where there are multiple such contexts within the management domain. Often, a context is a physical or logical device. A context is always defined as a subset of a single SNMP entity. To identify an individual item of management information within the management domain, its contextName and contextEngineID must be identified in addition to its object type and its instance. A default context is assumed when no context is specified.

A MIB object is usually a scalar object. A MIB object may have a tabular form with rows and columns. Such an object is composed of a sequence of rows, with each row composed of a sequence of typed values. The index is a subset (1-2 items) of the typed values in the row. An index value identifies the row in the table.

In SMI, a table is constructed as a SEQUENCE OF its entries. For example, the IpAddrTable from [RFC4293] has the following definition:
ipv6InterfaceTable OBJECT-TYPE
SYNTAX SEQUENCE OF Ipv6InterfaceEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "The table containing per-interface IPv6-specific information."
::= { ip 30 }

ipv6InterfaceEntry OBJECT-TYPE
SYNTAX Ipv6InterfaceEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION "An entry containing IPv6-specific information for a given interface."
INDEX { ipv6InterfaceIfIndex }
::= { ipv6InterfaceTable 1 }

Ipv6InterfaceEntry ::= SEQUENCE {
  ipv6InterfaceIfIndex        InterfaceIndex,
  ipv6InterfaceReasmMaxSize   Unsigned32,
  ipv6InterfaceIdentifier     Ipv6AddressIfIdentifierTC,
  ipv6InterfaceEnableStatus   INTEGER,
  ipv6InterfaceReachableTime  Unsigned32,
  ipv6InterfaceRetransmitTime Unsigned32,
  ipv6InterfaceForwarding     INTEGER
}

The descriptor (name) of the MIB table is used for the name of the CoMI variable. However, there is no explicit mention of the names "ipv6InterfaceEntry" and "Ipv6InterfaceEntry". Instead, the value of the main CoMI variable consists of an array, each element of which contains 7 CoMI variables: one element for "ipv6InterfaceIfIndex", one for "ipv6InterfaceReasmMaxSize" and so on until "ipv6InterfaceForwarding".

3.2. CoMI Function Set

Two types of interfaces are supported by CoMI:

single value  Reading/Writing one MIB variable, specified in the URI with path /mg/mib/descriptor or with path /mg/mib/OID.

multiple values  Reading writing arrays or multiple MIB variables, specified in the payload.
The examples in this section use a JSON payload with one or more entries describing the pair (descriptor, value), or (OID, value). The CBOR syntax of the payloads is specified in Section 4.

3.2.1. Single MIB values

A request to read the value of a MIB variable is sent with a confirmable CoAP GET message. The single MIB variable is specified in the URI path with the OID or descriptor suffixing the /mg/mib/ path name. When the descriptor is used to specify the MIB value, the same descriptor may be present in multiple module. To disambiguate the descriptor the "mod" uri-query attribute specifies the enveloping modules. A request to set the value of a MIB variable is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request.

TODO: for multicast send unconfirmed PUT

Using for example the same MIB from [RFC1213] as used in [RFC3416], a request is sent to retrieve the value of sysUpTime specified in module SNMPv2-MIB. The answer to the request returns a (descriptor, value) pair.

For clarity of the examples, in this and all following examples the payload is expressed in JSON, although the operational payload is specified to be in CBOR, as described in Section 4.

REQ: GET example.com/mg/mib/sysUpTime?mod=SNMPv2-MIB

RES: 2.05 Content (Content-Format: application/json)
{
    "sysUpTime" : 123456
}

Another way to express the descriptor of the required value is by specifying the pair (descriptor or oid, null value) in the payload of the request message.
REQ: GET example.com/mg/mib/(Content-Format: application/json)
{
    "SNMPv2-MIB.sysUpTime" : "null"
}

RES: 2.05 Content (Content-Format: application/json)
{
    "SNMPv2-MIB.sysUpTime" : 123456
}

The module name SNMPv2-MIB can be omitted when there is no possibility of ambiguity. The module.descriptor can of course be replaced with the corresponding oid.

In some cases it is necessary to determine the "context" by specifying a context name and a contextEngine identifier. The context can be specified in the URI with the uri-query attribute "con". Based on the example of figure 3 in section 3.3 of [RFC3411], the context name, bridge1, and the context Engine Identifier, 800002b804616263, separated by an underscore, are specified in the following example:

REQ: GET example.com/mg/mib/sysUPTime?con=bridge1_800002b804616263

RES: 2.05 Content (Content-Format: application/json)
{
    "sysUpTime" : 123456
}

The specified object can be a table. The returned payload is composed of all the rows associated with the table. Each row is returned as a set of (column name, value) pairs. For example the GET of the ipNetToMediaTable, sent by the managing entity, results in the following returned payload sent by the managed entity:
REQ: GET example.com/mg/mib/ipNetToMediaTable

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

```json
{
    "ipNetToMediaTable": [
        {
            "ipNetToMediaIfIndex": 1,
            "ipNetToMediaPhysAddress": "00:00::10:01:23:45",
            "ipNetToMediaNetAddress": "10.0.0.51",
            "ipNetToMediaType": "static"
        },
        {
            "ipNetToMediaIfIndex": 1,
            "ipNetToMediaPhysAddress": "00:00::10:54:32:10",
            "ipNetToMediaNetAddress": "9.2.3.4",
            "ipNetToMediaType": "dynamic"
        },
        {
            "ipNetToMediaIfIndex": 2,
            "ipNetToMediaPhysAddress": "00:00::10:98:76:54",
            "ipNetToMediaNetAddress": "10.0.0.15",
            "ipNetToMediaType": "dynamic"
        }
    ]
}
```

It is possible that the size of the returned payload is too large to fit in a single message.

CoMI gives the possibility to send the contents of the objects in several fragments with a maximum size. The "sz" link-format attribute [RFC6690] can be used to specify the expected maximum size of the mib resource in (identifier, value) pairs. The returned data MUST terminate with a complete (identifier, value) pair.

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.

3.2.2. multi MIB values

A request to read multiple MIB variables is done by expressing the pairs (MIB descriptor, null) in the payload of the GET request message. A request to set multiple MIB variables is done by expressing the pairs (MIB descriptor, null value) in the payload of
the PUT request message. The key word `_multiMIB` is used as array
name to signal that the payload contains multiple MIB values as
separate `_multiMIB` array entries.

The following example shows a request that specifies to return the
values of `sysUpTime` and `ipNetToMediaTable`:

REQ: GET example.com/mg/mib (Content-Format: application/json)
{
   "_multiMIB" : [
      { "sysUpTime" : "null"},
      { "ipNetToMediaTable" : "null" }
   ]
}

RES: 2.05 Content (Content-Format: application/json)
{
   "_multiMIB" : [
      { "sysUpTime" : 123456},
      { "ipNetToMediaTable" : [
          { "ipNetToMediaIfIndex" : 1,
            "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",
            "ipNetToMediaNetAddress" : "10.0.0.51",
            "ipNetToMediaType" : "static"
          },
          { "ipNetToMediaIfIndex" : 1,
            "ipNetToMediaPhysAddress" : "00:00::10:54:32:10",
            "ipNetToMediaNetAddress" : "9.2.3.4",
            "ipNetToMediaType" : "dynamic"
          },
          { "ipNetToMediaIfIndex" : 2,
            "ipNetToMediaPhysAddress" : "00:00::10:98:76:54",
            "ipNetToMediaNetAddress" : "10.0.0.15",
            "ipNetToMediaType" : "dynamic"
          }
      ]
   ]
}
3.2.3. Table row

The managing entity MAY be interested only in certain table entries. One way to specify a row is to specify its row number in the URI with the "row" uri-query attribute. The specification of row=1 returns row 1 values of the ipNetToMediaTable in the example:

REQ: GET example.com/mg/mib/ipNetToMediaTable?row=1

RES: 2.05 Content (Content-Format: application/json)
{
  "ipNetToMediaTable" : [
    {
      "ipNetToMediaIfIndex" : 1,
      "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",
      "ipNetToMediaNetAddress" : "10.0.0.51",
      "ipNetToMediaType" : "static"
    }
  ]
}

An alternative mode of selection is by specifying the value of the INDEX attributes. Towards this end, the managing entity can include the required entries in the payload of its "GET" request by specifying the values of the index attributes. The key word _indexMIB is used to specify the index value.

For example, to obtain a table entry from ipNetToMediaTable, the rows are specified by specifying the index attributes: ipNetToMediaIfIndex and ipNetToMediaNetAddress. The managing entity could have sent a GET with the following payload:
REQ: GET example.com/mg/mib/ipNetToMediaTable (Content-Format: application/json)

{ "_indexMIB" : 
  {  
    "ipNetToMediaIfIndex" : 1,  
    "ipNetToMediaNetAddress" : "9.2.3.4"  
  } 
}

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

{ "ipNetTOMediaTable" : [ 
  {  
    "ipNetToMediaIfIndex" : 1,  
    "ipNetToMediaPhysAddress" : "00:00::10:01:23:45",  
    "ipNetToMediaNetAddress" : "9.2.3.4",  
    "ipNetToMediaType" : "static"  
  } 
] } 

Constrained devices MAY support this kind of filtering. However, if they don’t support it, they MUST ignore the payload in the GET request and handle the message as if the payload was empty.

It is advised to keep MIBs for constrained entities as simple as possible, and therefore it would be best to avoid extensive tables.

TODO: Describe how the contents of the next lexicographical row can be returned.

3.2.4. Error returns

When a variable with the specified name cannot be processed, CoAP Error code 5.01 is returned. In addition, a MIB specific error can be returned in the payload as specified in Section 8.

4. Mapping SMI to CoMI payload

The SMI syntax is mapped to CBOR necessary for the transport of MIB data in the CoAP payload. This section first describes an additional data reduction technique by creating a table that maps string values to numbers used in CBOR encoded data.

The section continues by describing the mapping from SMI to CBOR. The mapping is inspired by the mapping from SMI to JSON via YANG [RFC6020], as described in [RFC6643] defining a mapping from SMI to
YANG, and [I-D.lhotka-netmod-yang-json] defining a mapping from YANG to JSON.

Notice that such conversion chain MAY be virtual only, as SMI could be converted directly to JSON by combining the rules from the above documents.

4.1. Mapping strings to CBOR

Because descriptors may be rather long and may occur repeatedly, CoMI allows for association of a string with an integer, henceforth called "string number". The association between string and string number is done through a translation table, leveraging CBOR encoding.

Using the notational convention from Appendix A, the CBOR data has the following syntax:

cBorMIB       : CBorMIB;

*CBorMIB {
   xlatTableID : uint;
   mibString   : map( uint, . );
}

The main structure consist of an array of two elements: "xlatTableID" and "mibString".

The values of the MIB strings are stored in the "mibString" field. This field consist of integer-value pairs. The integers correspond to the string numbers, whereas the values contain the actual value of the associated string.

The "xlatTableID" contains an integer that is used to indentify the translation table. The translation table can be acquired as follows:

GET /mg/xlat/[xlatTableID]

where "[xlatTableID]" is replaced by the the value of xlatId from the CBoRMIB structure, encoded as a hexadecimal integer without leading zeros.

The maintenance of the table is described in Section 7.2.

The use of the table is to initialize devices with the strings which will be frequently used, such as the strings of the descriptors in the MIB variables. The transmitted CBOR data will contain the string
numbers and not the entire descriptor strings, leading to appreciable
data reduction.

It is important that sender and receiver have identical versions of
the table.

The translation table is serialized to the payload in the following
fashion:

```c
xlatTable     : XLatTable;
*XLatTable {
    xlatId      : uint;
    xlatData    : map( uint, tstr );
}
```

where "xlatId" has the same value as "xlatId" in the CBorMIB
structure, and "xlatData" is a CBOR map between the string number and
associated variable descriptor.

4.2. Mapping SMI to CBOR

4.2.1. General overview

Starting from the intermediate conversion from SMI to YANG as defined
in [RFC6643], This section defines how to convert the resulting YANG
structure to CBOR [RFC7049]. The actual conversion code from SMI to
YANG and subsequently YANG to CBOR MAY be direct conversion code from
SMI to CBOR or a sequence of existing SMI to YANG conversion code
followed by YANG to CBOR conversion code.

4.2.2. 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.

```
<table>
<thead>
<tr>
<th>YANG type</th>
<th>CBOR type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>int8, int16,</td>
<td>unsigned int (major type 0)</td>
<td>The CBOR integer type depends on the sign of the actual value.</td>
</tr>
</tbody>
</table>
```

[Page 16]
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int32, int64, uint16, uint32, uint64, decimal64</td>
<td>or negative int (major type 1)</td>
</tr>
<tr>
<td>boolean</td>
<td>either &quot;true&quot; (major type 7, simple value 21) or &quot;false&quot; (major type 7, simple value 20)</td>
</tr>
<tr>
<td>string</td>
<td>text string (major type 3)</td>
</tr>
<tr>
<td>enumeration</td>
<td>unsigned int (major type 0)</td>
</tr>
<tr>
<td>bits</td>
<td>array of text strings</td>
</tr>
<tr>
<td>binary</td>
<td>byte string (major type 2)</td>
</tr>
<tr>
<td>empty</td>
<td>null (major type 7, simple value 22)</td>
</tr>
<tr>
<td>union</td>
<td>Similar ot the JSON transcription from [I-D.lhotka-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].</td>
</tr>
<tr>
<td>leaf-list</td>
<td>array (major type 4)</td>
</tr>
<tr>
<td>list</td>
<td>map (major type 4)</td>
</tr>
<tr>
<td>container</td>
<td>map (major type 4)</td>
</tr>
</tbody>
</table>

Each text string contains the name of a bit value that is set.

TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...

The array is encapsulated in the map associated with the descriptor.

Like the higher level map, the lower level map contains descriptor number - value pairs of the elements in the list.

The map contains descriptor
5) 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

4.2.3. Examples

4.2.3.1. ipNetToMediaTable to JSON/CBOR

The YANG translation of the SMI specifying the ipNetToMediaTable yields:

```yml
container ipNetToMediaTable {
    list ipNetToMediaEntry {
        leaf ipNetToMediaIfIndex {
            type: int32;
        }
        leaf ipNetToPhysAddress {
            type: phys-address;
        }
        leaf ipNetToMediaNetAddress {
            type: ipv4-address;
        }
        leaf ipNetToMediaType {
            type: int32;
        }
    }
}
```

The corresponding JSON looks like:
An example CBOR instance of the MIB can be found in Figure 1. The names "ipNetToMediaTable", "ipNetToMediaEntry", and "ipNetToMediaIfIndex" are represented with the string numbers 00, 01, and 02 as described in Section 4.1.

```
82                      # two element array
19 43 A1                # translation table ID 43A1
BF                      # indefinite length map
00                      # descriptor number related to
# ipNetToMediaTable
BF                      # indefinite length map related to
# ipNetToMediaTable
01                      # descriptor number related to
# ipNetToMediaTable
BF                      # map related to ipNetToMediaEntry
02                      # descriptor number associated with
# ipNetToMediaIfIndex
1A 00 00 00 01          # associated value as 32-bit integer
# ...
FF                      #
FF                      #
FF                      #
```

Figure 1: Example CBOR encoding for ifTable

The associated "descriptor string" to "string number" translation table is given in Figure 2.
4.2.4. 6LoWPAN MIB

A MIB for 6LoWPAN is defined in [I-D.schoenw-6lowpan-mib]. The
document also provides an example JSON representation in its
Appendix A. Figure 3 shows the associated CBOR representation with
string number, and Figure 4 shows the corresponding string to string
conversion table.

Figure 2: Translation table for ifTable
00  # 0
07  # "lowpanInReasmReqds"
16  # 22
08  # "lowpanInReasmFails"
02  # 02
09  # "lowpanInReasmOKs"
14  # 20
0A  # "lowpanInCompReqds"
10  # 16
0B  # "lowpanInCompFails"
02  # 2
0C  # "lowpanInCompOKs"
0E  # 14
0D  # "lowpanInDiscards"
01  # 01
0E  # "lowpanInDelivers"
0C  # 12
0F  # "lowpanOutRequests"
0C  # 12
10  # "lowpanOutCompReqds"
00  # 0
11  # "lowpanOutCompFails"
00  # 0
12  # "lowpanOutCompOKs"
00  # 0
13  # "lowpanOutFragReqds"
05  # 5
14  # "lowpanOutFragFails"
00  # 0
15  # "lowpanOutFragOKs"
05  # 5
16  # "lowpanOutFragCreates"
08  # 8
17  # "lowpanOutMeshHopLimitExceeds"
00  # 0
18 18  # "lowpanOutMeshNoRoutes"
00  # 0
18 19  # "lowpanOutMeshRequests"
00  # 0
18 1A  # "lowpanOutMeshForwds"
00  # 0
18 1B  # "lowpanOutMeshTransmits"
00  # 0
18 1C  # "lowpanOutDiscards"
00  # 0
18 1D  # "lowpanOutTransmits"
0F  # 15
FF
Figure 3: Example CBOR encoding for the 6LoWPAN MIB

82 1A 8B 47 88 F3 BF
# two element array # translation table ID 8B4788F3 # indefinite length map
00 75 # "LOWPAN-MIB:LOWPAN-MIB"
01 #
02 72 ... # "lowpanReasmTimeout"
03 70 ... # "lowpanInReceives"
04 71 ... # "lowpanInHdrErrors"
05 74 ... # "lowpanInMeshReceives"
06 72 ... # "lowpanInMeshForwds"
07 74 ... # "lowpanInMeshDelivers"
08 72 ... # "lowpanInReasmReqds"
09 72 ... # "lowpanInReasmFails"
0A 70 ... # "lowpanInReasmOKs"
0B 71 ... # "lowpanInCompReqds"
0C 71 ... # "lowpanInCompFails"
0D 6F ... # "lowpanInCompOKs"
0E 70 ... # "lowpanInDiscards"
0F 70 ... # "lowpanInDelivers"
10 72 ... # "lowpanOutRequests"
11 72 ... # "lowpanOutCompReqds"
12 72 ... # "lowpanOutCompFails"
13 70 ... # "lowpanOutCompOKs"
14 72 ... # "lowpanOutFragReqds"
Figure 4: Translation table for the 6LoWPAN MIB

In this example, a GET to /mg/mib/lowpanOutFragFails would give:

```
82                         # two element array
1A 8B 47 88 F3            # translation table ID 8B4788F3
BF                      # indefinite length map
14                    # "lowpanOutFragFails"
00                    # 0
FF                      #
```

5. MIB discovery

MIB objects are discovered like resources with the standard CoAP resource discovery. Performing a GET on "/.well-known/core" with rt=core.mg.mib returns all MIB descriptors and all OIDs which are available on this device. For table objects there is no further possibility to discover the row descriptors. For example, consider there are two MIB objects with descriptors "sysUpTime" and "ipNetToMediaTable" associated with OID 1.3.6.1.2.1.1.3 and 1.3.6.1.2.1.4.22.
The link format attribute 'oid' is used to associate the name of the
MIB resource with its OID. The OID is written as a string in its
conventional form.

Notice that a MIB variable normally is associated with a descriptor
and an OID. The OID is unique, whereas the descriptor is unique in
combination with the module name.

The "mod", "con", and "rt" attributes can be used to filter resource
queries as specified in [RFC6690].

6. Trap functions

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

TODO: Observe example

In the registration request, the managing entity 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 managing entity wants the
managed device to send the trap information to a multicast address.

7. MIB 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.

7.1. 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.

7.2. Conversion tables

POST is used to initialize a conversion table. At the arrival of the
POST, all existing tables are removed and new tables as specified by
the payload are created with the contents specified in the payload.
When the payload of the POST is empty, no table is created.

PUT is used to create new entries in an existing table and overwrite
existing entries. When the payload of the PUT contains a non
existing table, a new table with the new identity is created. When
the payload of the PUT contains a table with an already existing
identifier, two possibilities exist:

- exiting string value: the contents of the existing pair is
  overwritten with the pair in the payload.
- new string value: A new pair is created in the table with the pair in
  the payload.

8. 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:

```
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.
<table>
<thead>
<tr>
<th>CoMI Error Code</th>
<th>CoAP Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4.00</td>
<td>General error</td>
</tr>
<tr>
<td>1</td>
<td>4.00</td>
<td>Malformed CBOR data</td>
</tr>
<tr>
<td>2</td>
<td>4.00</td>
<td>Incorrect CBOR datatype</td>
</tr>
<tr>
<td>3</td>
<td>4.00</td>
<td>Unknown MIB variable</td>
</tr>
<tr>
<td>4</td>
<td>4.00</td>
<td>Unknown translation table</td>
</tr>
<tr>
<td>5</td>
<td>4.05</td>
<td>Attempt to write read-only variable</td>
</tr>
<tr>
<td>0..2</td>
<td>5.01</td>
<td>Access exceptions</td>
</tr>
<tr>
<td>0..18</td>
<td>5.00</td>
<td>SMI error status</td>
</tr>
</tbody>
</table>

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].

9. Security Considerations

TODO: follows CoAP security provisioning.

10. IANA Considerations

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

Content types to be registered:

- application/comi+json
- application/comi+cbor

11. 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. Juergen Schoenwalder has commented on inconsistencies and missing aspects of SNMP in earlier versions of the draft. The draft has benefited from comments by Thomas Watteyne, Dee Denteneer, Esko Dijk, and Michael van Hartskamp. The CBOR encoding borrows
extensively from Ladislav Lhotka’s description on conversion from YANG to JSON.

12. 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

13. References

13.1. Normative References


[I-D.becker-core-coap-sms-gprs]
13.2. Informative References


Appendix A. 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