core
Internet-Draft
Intended status: Informational
Expires: November 9, 2014

P. van der Stok
consultant
B. Greevenbosch
Huawei Technologies

May 8, 2014

CoAp Management Interfaces
draft-vanderstok-core-comi-04

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. The introduction motivates the choices of CoMI with respect to utilization of YANG, NETCONF, SMI, CBOR, CoAP, and URI structure.

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 November 9, 2014.

Table of Contents

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, whereas RESTCONF uses YANG encoded in JSON and HTTP/TCP to access NETCONF data stores. CoMI is more low resource oriented than RESTCONF is, and only supports the methods GET, PUT, POST and DELETE. RESTCONF also uses uses the HTTP methods HEAD, OPTIONS and PATCH, which are not available in CoAP.

Given the automatic conversion from SMI to YANG, from YANG to JSON, from YANG to XML and from JSON to CBOR, the transported data format is not strongly related to the chosen management protocol.

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 is 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 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. As such it is close to the device management of the Open Mobile Alliance described in [OMA]. However, the structure of the MIB does not reflect the structure of the OMA management objects. It is assumed that the structure and semantics of the management data are the most important aspect of a standard like the MIB. The right path forward to integrate OMA management with COMI is the adapatation of the MIB structure by OMA.

COMI supports the conversion of SMIv2 via YANG to CBOR. Assuming that CBOR is used for the tanport of NETCONF data, the utilization of the CBOR conversion table to reduce payload size can be envisaged for NETCONF data as well.

COMI uses a simple URI to access the MIB 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 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.
- conv with path /mg/conv and CBOR content format.

The mib resource provides access to the MIBs as described in Section 3.2. The conv resource provides access to a table that maps

a string to CBOR identifier, as described in Section 4.1. The mib and conv 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]:

name	path	RT	Data Type
Management	/mg	core.mg	n/a
MIB	/mg/mib	core.mg.mib	application/cbor
conv	/mg/conv	core.mg.conv	application/cbor

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:

- A data definition language that is referred to as Structure of Management Information (SMI)[RFC2578].
- The Management Information Base (MIB) which contains the information to be managed and is defined for each specific function to be managed [RFC3418].
- A protocol definition referred to as Simple Network Management Protocol (SNMP) [RFC3416].
- Security and administration that provides SNMP message based security on the basis of the user-based security model [RFC3414].
- 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:

- Get Request, transmits a list of OBJECT-IDENTIFIERs to be paired with values.
- GetNext Request, transmits a list of OBJECT-IDENTIFIERs to which lexicographic successors are returned for table traversal.
- GetBulk Request, transmits a list of OBJECT-IDENTIFIERs and the maximum number of expected paired values.
- Response, returns an error or the (OBJECT-IDENTIFIER, value) pairs for the OBJECT-IDENTIFIERs specified in Get, GetNext, GetBulk, Set, or Inform Requests.
- Set Request, transmits a list of (OBJECT-IDENTIFIERs, value) pairs to be set in the specified MIB object.
- Trap, sends an unconfirmed message with a list of (OBJECT-IDENTIFIERs, value) pairs to a notification requesting end-point.
- Inform Request, sends a confirmed message with a list of (OBJECT-IDENTIFIERs, value) pairs to a notification requesting end-point.

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, ipv6InterfaceTable, 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

Three 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.

Discovery: Discovery of MIB descriptors as specified in [RFC6690].

The examples in this section use a JSON payload with one or more entries describing the pair (descriptor, value), or (OID, value). CoMI transports the CBOR format to transport the equivalent contents. 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.

As announced, in all examples the payload is expressed in JSON, although the operational payload is specified to be in CBOR.

```
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)
{
   "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, 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. 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

```
REQ: GET example.com/.well-known/core?rt=core.mg.mib

RES: 2.05 Content (Content-Format: application/text)
</mg/mib/sysUpTime>;rt="core.mg.mib";oid="1.3.6.1.2.1.1.3";
  mod="SNMPv2-MIB"
</mg/mib/ipNetToMediaTable>;rt="core.mg.mib";oid="1.3.6.1.2.1.4.22";
  mod="ipMIB"
```

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].

3.2.5. 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 7.

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. CBOR format for MIB data

Because descriptors may be rather long and may occur repeatedly, CoMI allows for association of a given string value with an integer, henceforth called "string number". The association between string value and string number is done through a conversion table, leveraging CBOR encoding. Section 4.2 defines how the conversion table is generated.

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

```
cBorMIB       : CBorMIB;

*CBorMIB {
  convTableId : tstr;
  mibData     : map( uint, . );
}
```

The main structure consist of an array of two elements: "convTableId" and "mibData".

The conversion table identifier "convTableId" is constructed from the LAST_UPDATED attribute and module name as defined in Section 4.2.

The values of the MIB variables are stored in the "mibData" 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 MIB variable. Section 4.3 elaborates on the mapping from SMI to CBOR.

4.2. Table generation

The conversion table contents MUST be automatically generated from the MIB module as specified in this section. Automatic generation in the managed device removes the need to transport the tables and subsequently store them in the nodes, and avoids a cumbersome management of the conversion tables.

The conversion table identifier "convTableId" is generated from the LAST-UPDATED field in the MODULE IDENTITY macro, and the module name as follows:

```
convTableId = moduleName UNDERSCORE lastUpdateTimestamp
UNDERSCORE  = %x5F
```

where moduleName is the module name, and lastUpdateTimestamp is the value of the related LAST-UPDATED field in the MODULE-IDENTITY macro, using the format with four year digits as defined in [RFC2578].

The conversion table is generated using the following algorithm:

1. Collect all used OIDs/descriptors defined in the MIB object, and as well as those imported using IMPORTS.

2. Sort the collection of OIDs according to the following rules:

    1. x.y.z < x.y.z.k

    2. x.y.z... < x.y.k... if z<k

3. Assign the string numbers to the sorted list of OIDs, in numerical order starting with 1.

Note that variables imported with IMPORTS could be tables, such that the OIDs of the table entries need to be included in the collection of OIDs as well.

4.3. Mapping SMI to CBOR

4.3.1. 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 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.

4.3.2. 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.

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.lhotka-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 4)	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 decriptor 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

4.3.3. Examples

4.3.3.1. ipNetToMediaTable to JSON/CBOR

The YANG translation of the SMI specifying the ipNetToMediaTable yields:

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

The coresponding JSON looks like:

```
{
"ipNetToMediaTable" : {
    "ipNetToMediaEntry" : [
        {
        "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"
        }
    ]
    }
}
```

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
   74 49 50 2d 4d 49 42 5f
   32 30 30 36 30 32 30 32
   30 30 30 30 5a           # "IP-MIB_200602020000Z"
   BF                       # indefinite length map
     00                     # descriptor number related to
                            # ipNetToMediaTable
     BF                     # indefinite length map related to
                            # ipNetToMediaTable
        01                  # descriptor number related to
                            # ipNetToMediaEntry
        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" conversion table is given in Figure 2.

```
82                          # two element array
   74 49 50 2d 4d 49 42 5f
   32 30 30 36 30 32 30 32
   30 30 30 30 5a           # "IP-MIB_200602020000Z"
   BF                       # indefinite length map
      00                    # descriptor number related to
                            # ipNetToMediaTable
      72 69 70 50 65 74 57
      6F 51 65 64 61 57 61
      62 6C 65              # "ipNetToMediaTable"
      01                    # descriptor number related to
                            # ipNetToMediaEntry
      72 69 70 50 65 74 57
      6F 51 65 64 61 45 6E
      74 72 78                 # "ipNetToMediaEntry"
      02                    # descriptor number related to
                            # ipNetToMediaIfIndex
      75 69 70 50 65 74 57
      6F 51 65 64 61 61 49
      66 49 6E 64 65 77     # "ipNetToMediaIfIndex"
      # ...
   FF
```

Figure 2: Conversion table for ifTable

4.3.4. 6LoWPAN MIB

A MIB for 6LoWPAN is defined in [I-D.ietf-6lo-lowpan-mib] with an example JSON representation in its Appendix A. Appendix B.3 shows the associated CBOR representation with string number, and the corresponding string to string number conversion table.

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

```
82                          # two element array
   78 18 4c 4f 57 50 41 4e
   2d 4d 49 42 5f 32 30 31
   34 30 34 30 38 30 30 30
   30 5a                    # conversion table id:
                            # "LOWPAN-MIB_201404080000Z"
   BF                       # indefinite length map
     18 1B                  # "lowpanOutFragFails"
     00                     # 0
   FF
```

5. 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.

6. 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.

6.1. 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.

6.2. Conversion table management

Since the conversion table is unique for a MIB, it can be generated off-line by a managing entity, and independently generated in the managed device with the same result. Because different versions of a given MIB may be available the managing entity must be able to check the version on the managed device. The convTableId can be acquired with a confirmable GET request:

```
REQ: GET example.com/mg/conv/ident

RES: 2.05 Content (Content-Format: application/json)
{
    "convTableId" : convTableId
}
```

The conversion table MAY be available on a managed device, and can then be acquired as follows:

```
GET /mg/conv/[convTableId]
```

where "[convTableId]" is replaced by the the value of convTableId from the CBorMIB structure.

In case the managed device provides the table, it must be transmitted in the payload of the GET response using the following format:

```
convTable     : ConvTable;

*ConvTable {
  convTableId : tstr;
  convData    : map( uint, tstr );
}
```

where "convTableId" is the conversion table identifier, as defined in Section 4.2, and "convData" is a CBOR map between the string number and associated variable descriptor.

7. 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 readible 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 associted with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

8. 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 [I-D.ietf-core-coap]). 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.

9. IANA Considerations

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

Content types to be registered:

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

10. 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 (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.

11. 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

- o Definition of CBOR string conversion tables for data reduction
- o use of Block for multiple fragments
- o Error returns generalized
- o SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- o Added security considerations

Changes from version 03 to version 04

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

12. References

12.1. 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.

[I-D.becker-core-coap-sms-gprs] Becker, M., Li, K., Poetsch, T., and K. Kuladinithi, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-04 (work in progress), August 2013.

[I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-14 (work in progress), October 2013.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-13 (work in progress), April 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.lhotka-netmod-yang-json]
Lhotka, L., "Modeling JSON Text with YANG", draft-lhotka-netmod-yang-json-02 (work in progress), September 2013.

12.2. 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.

[I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-18 (work in progress), December 2013.

[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-01 (work in progress), April 2014.

[I-D.bierman-netconf-restconf]
Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-bierman-netconf-restconf-04 (work in progress), February 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, .

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 preceeded 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.

Appendix B. Example conversion table and instance for the LOWPAN MIB

The conversion table of the 6LoWPAN MIB [I-D.ietf-6lo-lowpan-mib] is generated as described in this section.

B.1. Generating the convTableId

The module name is "LOWPAN-MIB", and the LAST-UPDATED field in the MODULE-IDENTITY has the value "201404080000Z". Hence the convId has the value "LOWPAN-MIB_201404080000Z".

B.2. Generating the string numbers

The following table shows how the string numbers are associated with the related strings and object IDs.

Object ID	Descriptor	String

		number
1.3.6.1.2.1	mib-2	1
1.3.6.1.2.1.2.2.1.1	ifIndex	2
1.3.6.1.2.1.XXXX	lowpanMib	3
1.3.6.1.2.1.XXXX.0	lowpanNotifications	4
1.3.6.1.2.1.XXXX.1	lowpanObjects	5
1.3.6.1.2.1.XXXX.2	lowpanConformance	6
1.3.6.1.2.1.XXXX.1.1	lowpanStats	7
1.3.6.1.2.1.XXXX.1.1.1	lowpanReasmTimeout	8
1.3.6.1.2.1.XXXX.1.1.2	lowpanInReceives	9
1.3.6.1.2.1.XXXX.1.1.3	lowpanInHdrErrors	10
1.3.6.1.2.1.XXXX.1.1.4	lowpanInMeshReceives	11
1.3.6.1.2.1.XXXX.1.1.5	lowpanInMeshForwds	12
1.3.6.1.2.1.XXXX.1.1.6	lowpanInMeshDelivers	13
1.3.6.1.2.1.XXXX.1.1.7	lowpanInReasmReqds	14
1.3.6.1.2.1.XXXX.1.1.8	lowpanInReasmFails	15
1.3.6.1.2.1.XXXX.1.1.9	lowpanInReasmOKs	16
1.3.6.1.2.1.XXXX.1.1.10	lowpanInCompReqds	17
1.3.6.1.2.1.XXXX.1.1.11	lowpanInCompFails	18
1.3.6.1.2.1.XXXX.1.1.12	lowpanInCompOKs	19
1.3.6.1.2.1.XXXX.1.1.13	lowpanInDiscards	20
1.3.6.1.2.1.XXXX.1.1.14	lowpanInDelivers	21
1.3.6.1.2.1.XXXX.1.1.15	lowpanOutRequests	22
1.3.6.1.2.1.XXXX.1.1.16	lowpanOutCompReqds	23

1.3.6.1.2.1.XXXX.1.1.17	lowpanOutCompFails	24
1.3.6.1.2.1.XXXX.1.1.18	lowpanOutCompOKs	25
1.3.6.1.2.1.XXXX.1.1.19	lowpanOutFragReqds	26
1.3.6.1.2.1.XXXX.1.1.20	lowpanOutFragFails	27
1.3.6.1.2.1.XXXX.1.1.21	lowpanOutFragOKs	28
1.3.6.1.2.1.XXXX.1.1.22	lowpanOutFragCreates	29
1.3.6.1.2.1.XXXX.1.1.23	lowpanOutMeshHopLimitExceeds	30
1.3.6.1.2.1.XXXX.1.1.24	lowpanOutMeshNoRoutes	31
1.3.6.1.2.1.XXXX.1.1.25	lowpanOutMeshRequests	32
1.3.6.1.2.1.XXXX.1.1.26	lowpanOutMeshForwds	33
1.3.6.1.2.1.XXXX.1.1.27	lowpanOutMeshTransmits	34
1.3.6.1.2.1.XXXX.1.1.28	lowpanOutDiscards	35
1.3.6.1.2.1.XXXX.1.1.29	lowpanOutTransmits	36
1.3.6.1.2.1.XXXX.1.2	lowpanIfStatsTable	37
1.3.6.1.2.1.XXXX.1.2.1	lowpanIfStatsEntry	38
1.3.6.1.2.1.XXXX.1.2.1.1	lowpanIfReasmTimeout	39
1.3.6.1.2.1.XXXX.1.2.1.2	lowpanIfInReceives	40
1.3.6.1.2.1.XXXX.1.2.1.3	lowpanIfInHdrErrors	41
1.3.6.1.2.1.XXXX.1.2.1.4	lowpanIfInMeshReceives	42
1.3.6.1.2.1.XXXX.1.2.1.5	lowpanIfInMeshForwds	43
1.3.6.1.2.1.XXXX.1.2.1.6	lowpanIfInMeshDelivers	44
1.3.6.1.2.1.XXXX.1.2.1.7	lowpanIfInReasmReqds	45
1.3.6.1.2.1.XXXX.1.2.1.8	lowpanIfInReasmFails	46
1.3.6.1.2.1.XXXX.1.2.1.9	lowpanIfInReasmOKs	47

1.3.6.1.2.1.XXXX.1.2.1.10	lowpanIfInCompReqds	48
1.3.6.1.2.1.XXXX.1.2.1.11	lowpanIfInCompFails	49
1.3.6.1.2.1.XXXX.1.2.1.12	lowpanIfInCompOKs	50
1.3.6.1.2.1.XXXX.1.2.1.13	lowpanIfInDiscards	51
1.3.6.1.2.1.XXXX.1.2.1.14	lowpanIfInDelivers	52
1.3.6.1.2.1.XXXX.1.2.1.15	lowpanIfOutRequests	53
1.3.6.1.2.1.XXXX.1.2.1.16	lowpanIfOutCompReqds	54
1.3.6.1.2.1.XXXX.1.2.1.17	lowpanIfOutCompFails	55
1.3.6.1.2.1.XXXX.1.2.1.18	lowpanIfOutCompOKs	56
1.3.6.1.2.1.XXXX.1.2.1.19	lowpanIfOutFraqRegds	57
1.3.6.1.2.1.XXXX.1.2.1.20	lowpanIfOutFragFails	58
1.3.6.1.2.1.XXXX.1.2.1.21	lowpanIfOutFragOKs	59
1.3.6.1.2.1.XXXX.1.2.1.22	lowpanIfOutFragCreates	60
1.3.6.1.2.1.XXXX.1.2.1.23	lowpanIfOutMeshHopLimitExceeds	61
1.3.6.1.2.1.XXXX.1.2.1.24	lowpanIfOutMeshNoRoutes	62
1.3.6.1.2.1.XXXX.1.2.1.25	lowpanIfOutMeshRequests	63

1.3.6.1.2.1.XXXX.1.2.1.26	lowpanIfOutMeshForwds	64
1.3.6.1.2.1.XXXX.1.2.1.27	lowpanIfOutMeshTransmits	65
1.3.6.1.2.1.XXXX.1.2.1.28	lowpanIfOutDiscards	66
1.3.6.1.2.1.XXXX.1.2.1.29	lowpanIfOutTransmits	67
1.3.6.1.2.1.XXXX.2.1	lowpanGroups	68
1.3.6.1.2.1.XXXX.2.1.1	lowpanStatsGroup	69
1.3.6.1.2.1.XXXX.2.1.2	lowpanStatsMeshGroup	70
1.3.6.1.2.1.XXXX.2.1.3	lowpanIfStatsGroup	71
1.3.6.1.2.1.XXXX.2.1.4	lowpanIfStatsMeshGroup	72
1.3.6.1.2.1.XXXX.2.2	lowpanCompliances	73
1.3.6.1.2.1.XXXX.2.2.1	lowpanCompliance	74

Notes:

- The IMPORTS "MODULE-IDENTITY", "OBJECT-TYPE", "Unsigned32", "Counter32", "OBJECT-GROUP" and "MODULE-COMPLIANCE" do not have associated Object IDs, and hence do not show up in the table.

- The OBJECT-GROUPs lowpanStatsGroup, lowpanIfStatsGroup, lowpanIfStatsMeshGroup cannot appear in the CBOR instance, as they are lost in the intermediate MIB to YANG conversion step. However, as they have been defined in the original MIB, they still appear in the conversion table.

- (FOR THE EDITOR:) the value XXXX is defined to be the RFC number once [I-D.ietf-6lo-lowpan-mib] becomes an RFC. It is logically greater than 7000.

B.3. Example instance

A MIB for 6LoWPAN is defined in [I-D.ietf-6lo-lowpan-mib]. The document also provides an example JSON representation in its

Appendix A. Figure 3 shows the associated CBOR representation with string number, using the string numbers derived in Appendix B.2.

The request would have been as follows:

```
GET /mg/mib/lowpanIfStatsTable
```

The resulting table would be as follows:

```
82                          # two element array
   78 18 4c 4f 57 50 41 4e
   2d 4d 49 42 5f 32 30 31
   34 30 34 30 38 30 30 30
   30 5a                    # conversion table id:
                            # "LOWPAN-MIB_201404080000Z"
   BF                       # indefinite length map
     18 25                  # "lowpanIfStatsTable"
     BF                     # indefinite length map related
                            # to lowpanIfStatsTable
        18 27               # "lowpanIfReasmTimeout"
        14                  # 20
        18 28               # "lowpanIfInReceives"
        18 2A               # 42
        18 29               # "lowpanIfInHdrErrors"
        00                  # 0
        18 2A               # "lowpanIfInMeshReceives"
        08                  # 8
        18 2B               # "lowpanIfInMeshForwds"
        00                  # 0
        18 2C               # "lowpanIfInMeshDelivers"
        00                  # 0
        18 2D               # "lowpanIfInReasmReqds"
        16                  # 22
        18 2E               # "lowpanIfInReasmFails"
        02                  # 02
        18 2F               # "lowpanIfInReasmOKs"
        14                  # 20
        18 30               # "lowpanIfInCompReqds"
        10                  # 16
        18 31               # "lowpanIfInCompFails"
        02                  # 2
        18 32               # "lowpanIfInCompOKs"
        0E                  # 14
        18 33               # "lowpanIfInDiscards"
        01                  # 01
        18 34               # "lowpanIfInDelivers"
        0C                  # 12
        18 35               # "lowpanIfOutRequests"
```

```
            0C                  # 12
            18 36               # "lowpanIfOutCompReqds"
            00                  # 0
            18 37               # "lowpanIfOutCompFails"
            00                  # 0
            18 38               # "lowpanIfOutCompOKs"
            00                  # 0
            18 39               # "lowpanIfOutFragReqds"
            05                  # 5
            18 3A               # "lowpanIfOutFragFails"
            00                  # 0
            18 3B               # "lowpanIfOutFragOKs"
            05                  # 5
            18 3C               # "lowpanIfOutFragCreates"
            08                  # 8
            18 3D               # "lowpanIfOutMeshHopLimitExceeds"
            00                  # 0
            18 3E               # "lowpanIfOutMeshNoRoutes"
            00                  # 0
            18 3F               # "lowpanIfOutMeshRequests"
            00                  # 0
            18 40               # "lowpanIfOutMeshForwds"
            00                  # 0
            18 41               # "lowpanIfOutMeshTransmits"
            00                  # 0
            18 42               # "lowpanIfOutDiscards"
            00                  # 0
            18 43               # "lowpanIfOutTransmits"
            0F                  # 15
          FF
        FF
```

Figure 3: Example CBOR encoding for the 6LoWPAN MIB

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