core
Internet-Draft
Intended status: Informational
Expires: April 20, 2014
P. van der Stok
consultant
B. Greevenbosch
Huawei Technologies
October 17, 2013

CoAp Management Interfaces
draft-vanderstok-core-comi-01

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 JSON, CBOR or XML (EXI). The mapping from SMI to XML, JSON and CBOR is specified.

Note

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

Status of This Memo

Copyright Notice

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 are necessary when small devices from different manufacturers are managed by applications originating from another set of 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].

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 JSON [JSON], CBOR [I-D.bormann-cbor] or XML [XML]. CoMI is intended for small devices. The XML or JSON overhead can be prohibitive. It is therefore recommended to transport CBOR or EXI [EXI] in the payload. CBOR, like BER used for SNMP, transports the data type in the payload. EXI uses a schema file that provides information about transported data types.

In [EXI-measurement] it is shown that EXI can be an order of magnitude smaller than the equivalent XML representation. Actually, the EXI structure adds the overhead per data unit of an EXI event (indicates the type of the following XML element) with a size that depends on the number of EXI event types present in the schema and its frequency of occurrence. In [JSON-XML] it is shown that memory and CPU usage for sending JSON encoded or XML encoded objects led on average to a 50% lower resource usage for JSON. Consequently, from a resource utilization point of view EXI and CBOR seem the right choice. CBOR does not need a schema, but EXI requires one for decoding. Consequently, for EXI different schema versions must be handled by a versioning scheme.

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 one sub-resource accessible with the path:

- MIB with path /mg/mib and an XML (EXI), JSON or CBOR content format.

The mib resource provides access to the MIBs described in Section 3.2. The mib resource is introduced as a sub resource to mg to permit later additions to CoMI mg resource.

XML and JSON schemas describe the structure of the MIBs. It is expected that given the verbosity of XML and JSON, CoMI messages will mostly use EXI or CBOR. 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/json
MIB	/mg/mib	core.mg.mib	application/cbor
MIB	/mg/mib	core.mg.mib	application/xml
MIB	/mg/mib	core.mg.mib	application/exi

The value of the EXI events have a different meaning dependent on the accompanying schema file. Schemas are sent to the device at initialization such that it can prepare the parsing tables in advance. Schema management is discussed in Section 7. No such schema is needed for 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 PDUs and their transport.

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].

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.

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

- o Trap, sends an unconfirmed message with a list of (OBJECT-IDENTIFIERs, value) pairs to a notification requesting end-point.

- o Inform Request, sends a confirmed message with a list of (OBJECT-IDENTIFIERs, value) pairs to a notification requesting end-point.

The binding of the notification to a destination is discussed in Section 6.

3.1.2. MIB structure

A MIB module is composed of MIB objects. 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. One descriptor can be related to several OIDs.

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. 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 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.

A MIB object has a descriptor and an OID that identifies it uniquely within the device. MIB objects are standardized by the IETF or by other relevant Standards Developing Organizations (SDO).

The examples in this section use a payload with one or more entries describing the pair (descriptor, value), or (OID, value). The 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. A request to set a value is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request.

Using for example the same MIB from [RFC1213] as used in [RFC3416], a request is sent to retrieve the value of sysUpTime. The answer to the request returns a (descriptor, value) pair. The syntax of the payload is specified in Section 4.

```
REQ: GET example.com/mg/mib/sysUpTime

RES: 2.05 Content (Content-Format: application/xxxx)
(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 a 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/xxxx)
    ((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"))
```

The used syntax is for demonstration purposes only. Rows are encapsulated witin brackets, similar to the individual row elements. The syntax of the payload is specified in Section 4.

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. The sequel can be asked by sending the same request but with a uri-query indicating the offset of the first of the next requested pairs. This offset is equal to the already received number of pairs. The uri-query has the form "of=" (without the quotes) followed by the offset as an integer in text format.

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.

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/xxxx)
    (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 atttributes. 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.

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
     (ipNetToMediaIfIndex , 1)
    (ipNetToMediaNetAddress, "9.2.3.4")

RES: 2.05 Content (Content-Format: application/xxxx)
    (ipNetToMediaIfIndex , 1)
```

```
     (ipNetToMediaPhysAddress , "00:00::10:54:32:10")
     (ipNetToMediaNetAddress, "9.2.3.4")
     (ipNetToMediaType , "dynamic")
```

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.

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. Two types of error code can be returned: exception or error-status as specified in Appendix A, according to the rules of [RFC3416]. For example when MIB "variable" does not exist:

```
REQ: GET example.com/mg/mib/variable

RES: 5.01 Not Implemented (Content-Format: application/xxx)
(exception, nosuchobject)
```

4. Mapping SMI to CoMI payload

The SMI syntax is mapped to XML, CBOR, and JSON syntax, necessary for the transport of MIB data in the CoAP payload.

4.1. Mapping SMI to JSON

MIB information can be stored in JSON through a JSON object containing a set of Name-Value pairs, where each pair has the following syntax:

```
Name : Value
```

"Name" is a string that either contains the variable descriptor from the SMI definition, or the OID of the associated variable. In the latter case, the "Name" string has the following ABNF syntax, where we borrow some ABNF definitions from [I-D.ietf-json-rfc4627bis]:

```
descriptor = quotation-mark oid-prefix int *(underscore int)
             quotation-mark
```

```
oid-prefix = %x6f %x69 %x64 underscore                        ; 'oid_'

underscore = %x5f                                             ; '_'
```

In other words, when "Name" contains an OID, it is represented as a JSON string in which the dots of the OID are replaced by underscores, and the OID is prefixed by "oid_".

The "Value" field contains the variable value, using the JSON data type as indicated in Table 1.

SMI type	JSON type	Specification
OBJECT-IDENTIFIER	array of integers	
Integer32	integer	
INTEGER	integer	
OCTET-STRING	Base64 encoded string	
BITS	array of integers	The integers can only take the values 0 and 1.
IpAddress	string	[RFC4291]
Counter32	integer	
Gauge32	integer	
TimeTicks	integer	
Counter64	integer	
Unsigned32	integer	
Table	array	Section 4.1.1

Table 1: Conversion of SMI types to JSON

For example, the number of UDP datagrams received can be obtained through the udpInDatagrams variable as specified in [RFC4113]. It could have the following outcome:

```
{
  "udpInDatagrams" : 82174
}
```

Alternatively, when the response uses an OID instead of the variable identifier, the outcome would be as follows:

```
{
  "oid_1_3_6_1_2_1_7_1" : 82174
}
```

Another example of a JSON encoding of two (fictional) MIB variables "aBitString" and "anObjectID", with types BITS and OBJECT-IDENTIFIER, is as follows:

```
{
  "aBitString" : [ 0, 1, 0, 0, 1, 1 ],
  "anObjectID" : [ 1, 3, 6, 1, 10 ]
}
```

4.1.1. Tables in JSON

If a MIB object is a table, it is represented as an array of rows, the elements of which contain tuples related to the rows.

For example, an udpEndpointTable would be encoded as follows:

```
{
  "udpEndpointTable" : [
    {
      "udpEndpointLocalAddressType" : 2,
      "udpEndpointLocalAddress" : "2001:D3B8::417A",
      "udpEndpointLocalPort" : 5683,
      "udpEndpointRemoteAddressType" : 2,
      "udpEndpointRemoteAddress" : "2001:5C3D::59C1",
      "udpEndpointRemotePort" : 8000,
      "udpEndpointInstance" : 14789215,
      "udpEndpointProcess" : 61572493
    },
    {
      "udpEndpointLocalAddressType" : 2,
      "udpEndpointLocalAddress" : "2001:D3B8::417A",
      "udpEndpointLocalPort" : 5683,
      "udpEndpointRemoteAddressType" : 2,
```

```
        "udpEndpointRemoteAddress" : "4301:5338::DFC1",
        "udpEndpointRemotePort" : 6000,
        "udpEndpointInstance" : 14892714,
        "udpEndpointProcess" : 157491
      }
    ]
  }
```

4.2. Mapping SMI to CBOR

CBOR [I-D.bormann-cbor] is a binary format designated for the transmission of structured information. This section discusses how CBOR can be used to exchange MIB data.

The MIB variables are represented in a CBOR map of indefinite length. Such a map is a list of paired data fields. The first data field of each pair contains either a string containing the name of the MIB variable, or an array of integers containing the MIB variable's OID. The second datafield contains the variable's value.

Table 2 gives an overview of the conversion from SMI to CBOR. If the first element of a pair in the map contains an OID, it uses the same CBOR format for the first element as defined in the table in the "OBJECT-IDENTIFIER" row.

SMI type	CBOR type	Specification
OBJECT-IDENTIFIER	array of unsigned integers	First element corresponds to the most left number in the OID.
Integer32	unsigned integer or negative integer	
INTEGER	unsigned integer or negative integer	
OCTET-STRING	byte string	
BITS	array of unsigned integers	The integers can only take the values 0 and 1.

IpAddress	text string	[RFC4291]
Counter32	unsigned integer	
Gauge32	unsigned integer	
TimeTicks	unsigned integer	
Counter64	unsigned integer	
Unsigned32	unsigned integer	
Table	array of maps	Section 4.2.1

Table 2: Conversion of SMI types to CBOR

For example, the variable "udpInDatagrams" of SMI type "Counter32" and value "82174" is reflected in CBOR as follows:

```
BF                                              # map (indefinite)
   6E 75 64 70 49 6E 44 61 74 61 67 72 61 6D 73 # "udpInDatagrams"
   1A 00 01 40 FE                               # 32-bit int 82174
FF                                              # end of map
```

Or alternatively, using the OID of "udpInDatagrams":

```
BF                                              # map (indefinite)
   88 01 03 06 01 02 01 07 01                   # 1.3.6.1.2.1.7.1
   1A 00 01 40 FE                               # 32-bit int 82174
FF                                              # end of map
```

4.2.1. Tables in CBOR

In case a MIB variable is a table, it is represented in CBOR with a indefinite length array of maps.

The array corresponds to the rows in the MIB table, and its elements consist of maps corresponding to the elements of each row. The encoding of the maps has the same syntax as the main map.

For example, the "udpEndpointTable" from Section 3.1.2 would be encoded in CBOR as follows:

```
BF                                 # main map (indefinite)
   70 75 64 70 45 6E 64 70 6F 69 6E
   74 54 61 62 6C 65               # "udpEndpointTable"
   9F                              # array (indefinite)
      BF                           # map for first row
         78 1B 75 64 70 45 6E 64 70
         6F 69 6E 74 4C 6F 63 61 6C
         41 64 64 72 65 73 73 54 79
         70 65                     # "udpEndpointLocalAddressType"
         02                        # integer 2
         77 75 64 70 45 6E 64 70 6f
         69 6e 74 4C 6f 63 61 6C 41
         64 64 72 65 73 73         # "udpEndpointLocalAddress"
         6f 32 30 30 31 3A 44 33 42
         38 3A 3A 34 31 37 41      # "2001:D3B8::417A"
         74 75 64 70 45 6E 64 70 6F
         69 6E 74 4C 6f 63 61 6C 50
         6F 72 74                  # "udpEndpointLocalPort"
         19 16 33                  # 5683
         78 1C 75 64 70 45 6E 64 70
         6F 69 6E 74 52 65 6D 6F 74
         65 41 64 64 72 65 73 73 54
         79 70 65                  # "udpEndpointRemoteAddressType"
         02                        # 2
         78 18 75 64 70 45 6E 64 70
         6F 69 6E 74 52 65 6D 6F 74
         65 41 64 64 72 65 73 73   # "udpEndpointRemoteAddress"
         6F 32 30 30 31 3A 35 43 33
         44 3A 3A 35 39 43 31      # "2001:5C3D::59C1"
         75 75 64 70 45 6E 64 70 6F
         69 6E 74 52 65 6D 6F 74 65
         50 6F 72 74               # "udpEndpointRemotePort"
         19 1F 40                  # 8000
         73 75 64 70 45 6E 64 70 6F
         69 6E 74 49 6E 73 74 61 6E
         63 65                     # "udpEndpointInstance"
         1A 00 E1 AA 5F            # 14789215
         72 75 64 70 45 6E 64 70 6F
         69 6E 74 50 72 6F 63 65 73
         73                        # "udpEndpointProcess"
         1A 03 AB 85 8D            # 61572493
      FF                           # end of first row
      BF                           # map for second row
         78 1B 75 64 70 45 6E 64 70
         6F 69 6E 74 4C 6F 63 61 6C
```

```
            41 64 64 72 65 73 73 54 79
            70 65                      # "udpEndpointLocalAddressType"
            02                         # integer 2
            77 75 64 70 45 6E 64 70 6F
            69 6E 74 4C 6F 63 61 6C 41
            64 64 72 65 73 73          # "udpEndpointLocalAddress"
            6F 32 30 30 31 3A 44 33 42
            38 3A 3A 34 31 37 41       # "2001:D3B8::417A"
            74 75 64 70 45 6E 64 70 6F
            69 6E 74 4C 6F 63 61 6C 50
            6F 72 74                   # "udpEndpointLocalPort"
            19 16 33                   # 5683
            78 1C 75 64 70 45 6E 64 70
            6F 69 6E 74 52 65 6D 6F 74
            65 41 64 64 72 65 73 73 54
            79 70 65                   # "udpEndpointRemoteAddressType"
            02                         # 2
            78 18 75 64 70 45 6E 64 70
            6F 69 6E 74 52 65 6D 6F 74
            65 41 64 64 72 65 73 73    # "udpEndpointRemoteAddress"
            6F 34 33 30 31 3A 35 33 33
            38 3A 3A 44 46 43 31       # "4301:5338::DFC1"
            75 75 64 70 45 6E 64 70 6F
            69 6E 74 52 65 6D 6F 74 65
            50 6F 72 74                # "udpEndpointRemotePort"
            19 17 70                   # 6000
            73 75 64 70 45 6E 64 70 6F
            69 6E 74 49 6E 73 74 61 6E
            63 65                      # "udpEndpointInstance"
            1A 00 E3 3E AA             # 14892714
            72 75 64 70 45 6E 64 70 6F
            69 6E 74 50 72 6F 63 65 73
            73                         # "udpEndpointProcess"
            1A 00 02 67 33             # 157491
         FF                            # end of second row
      FF                               # end of array
   FF                                  # end of main map
```

TBD: we may consider splitting up the CBOR definition in two parts, the first part containing a translation table with (integer, descriptor/oid) pairs, the second as above but using instead of the descriptor/oid the integers defined in the translation table. This would increase coding efficiency, but requires some extra work especially on the client (managing entity) side. This suggestion would lead to the following CBOR data:

```
   82                                  # two element array
```

```
      BF                                # translation map (indefinite)
         00                             # 0:
            70 75 64 70 45 6E 64 70 6F
            69 6E 74 54 61 62 6C 65     # "udpEndpointTable"
         01                             # 1:
            78 1B 75 64 70 45 6E 64 70
            6F 69 6E 74 4C 6F 63 61 6C
            41 64 64 72 65 73 73 54 79
            70 65                       # "udpEndpointLocalAddressType"
         02                             # 2:
            77 75 64 70 45 6E 64 70 6f
            69 6e 74 4C 6f 63 61 6C 41
            64 64 72 65 73 73           # "udpEndpointLocalAddress"
         03                             # 3:
            74 75 64 70 45 6E 64 70 6F
            69 6E 74 4C 6f 63 61 6C 50
            6F 72 74                    # "udpEndpointLocalPort"
         04                             # 4:
            78 1C 75 64 70 45 6E 64 70
            6F 69 6E 74 52 65 6D 6F 74
            65 41 64 64 72 65 73 73 54
            79 70 65                    # "udpEndpointRemoteAddressType"
         05                             # 5:
            78 18 75 64 70 45 6E 64 70
            6F 69 6E 74 52 65 6D 6F 74
            65 41 64 64 72 65 73 73     # "udpEndpointRemoteAddress"
         06                             # 6:
            75 75 64 70 45 6E 64 70 6F
            69 6E 74 52 65 6D 6F 74 65
            50 6F 72 74                 # "udpEndpointRemotePort"
         07                             # 7:
            73 75 64 70 45 6E 64 70 6F
            69 6E 74 49 6E 73 74 61 6E
            63 65                       # "udpEndpointInstance"
         08                             # 8:
            72 75 64 70 45 6E 64 70 6F
            69 6E 74 50 72 6F 63 65 73
            73                          # "udpEndpointProcess"
      FF
      BF                                # variables map (indefinite)
         00                             # 0="udpEndpointTable"
         9F                             # array (indefinite)
            BF                          # map for first row
               01                       # 1="udpEndpointLocalAddressType"
               02                       # integer 2
               02                       # 2="udpEndpointLocalAddress"
               6f 32 30 30 31 3A 44 33
               42 38 3A 3A 34 31 37 41 # "2001:D3B8::417A"
```

```
               03                     # 3="udpEndpointLocalPort"
               19 16 33               # 5683
               04                     # 4="udpEndpointRemoteAddressType"
               02                     # 2
               05                     # 5="udpEndpointRemoteAddress"
               6F 32 30 30 31 3A 35 43
               33 44 3A 3A 35 39 43 31 # "2001:5C3D::59C1"
               06                     # 6="udpEndpointRemotePort"
               19 1F 40               # 8000
               07                     # 7="udpEndpointInstance"
               1A 00 E1 AA 5F         # 14789215
               08                     # 8="udpEndpointProcess"
               1A 03 AB 85 8D         # 61572493
            FF                        # end of first row
            BF                        # map for second row
               01                     # 1="udpEndpointLocalAddressType"
               02                     # integer 2
               02                     # 2="udpEndpointLocalAddress"
               6F 32 30 30 31 3A 44 33
               42 38 3A 3A 34 31 37 41 # "2001:D3B8::417A"
               03                     # 3="udpEndpointLocalPort"
               19 16 33               # 5683
               04                     # 4="udpEndpointRemoteAddressType"
               02                     # 2
               05                     # 5="udpEndpointRemoteAddress"
               6F 34 33 30 31 3A 35 33
               33 38 3A 3A 44 46 43 31 # "4301:5338::DFC1"
               06                     # 6="udpEndpointRemotePort"
               19 17 70               # 6000
               07                     # 7="udpEndpointInstance"
               1A 00 E3 3E AA         # 14892714
               08                     # 8="udpEndpointProcess"
               1A 00 02 67 33         # 157491
            FF                        # end of second row
         FF                           # end of array
      FF                              # end of main map
```

Obviously this leads to less bytes to transmit, but it needs to define an extra translation table. The server can include the fixed mapping as a macro, so it does not need to calculate it for each request. In this example the translation map is for the whole structure. The map can be restricted to the entries that are used.

4.3. Mapping SMI to XML

The types specified by SMI can be represented by a XML syntax according to the specification in this section. The XML syntax is taken over from http://www.w3c.org/2001/XMLSchema-datatypes.

SMI type	XML type	Specification
OBJECT-IDENTIFIER:	array of int	
Integer32:	int	
INTEGER:	int	
OCTET-STRING:	Base64 encoded string	
BITS:	bits	Appendix A
IPAddress:	string	[RFC4291]
Counter32:	nonNegativeInteger	
Gauge32:	nonNegativeInteger	
TimeTicks:	time	
Counter64:	unsignedlong	
Unsigned32:	unsignedInt	
Table	MIBTable	Appendix A

Table 3: Conversion of SMI types to XML

4.3.1. Tables in XML

In case a MIB variable is a table, it is represented in XML with an indefinite length SEQUENCE of type entry. Each element of entry has a name and a choice of XML types. The associated schema is shown in Appendix A. For example the udpEndPointTable would look like:

```
<MIBTable>
  <MIBidentifier>
     <descriptor>"udpEndpointTable"</descriptor>
  </MIBidentifier>
  <row>
     <entry name="udpEndpointLocalAddressType" value="2"/>
    <entry name="udpEndpointLocalAddress" value="2001:D3B8::417A"/>
```

```
      <entry name="udpEndpointLocalPort" value="5683"/>
  <entry name="udpEndpointRemoteAddressType" value="2"/>
      <entry name="udpEndpointRemoteAddress" value="2001:5C3D::59C1"/>
      <entry name="udpEndpointRemotePort" value="8000"/>
      <entry name="udpEndpointInstance" value="14789215"/>
      <entry name="udpEndpointProcess" value="61572493"/>
    </row>
    <row>
      <entry name="udpEndpointLocalAddressType" value="2"/>
      <entry name="udpEndpointLocalAddress" value="2001:D3B8::417A"/>
      <entry name="udpEndpointLocalPort" value="5683"/>
      <entry name="udpEndpointRemoteAddressType" value="2"/>
      <entry name="udpEndpointRemoteAddress" value="4301:5338::DFC1"/>
      <entry name="udpEndpointRemotePort" value="6000"/>
      <entry name="udpEndpointInstance" value="147892714"/>
      <entry name="udpEndpointProcess" value="157491"/>
    </row>
  </MIBTable>
```

When the MIBidentifier is an OID the syntax for MIB object 1.3.6.1.2.1.1.3 looks like:

```
<MIBObject>
  <MIBidentifier>
     <oid>
       <ident>1</ident>
       <ident>3</ident>
       <ident>6</ident>
       <ident>1</ident>
       <ident>2</ident>
       <ident>1</ident>
       <ident>1</ident>
       <ident>3</ident>
    </oid>
  </MIBidentifier>
  <MIBvalue value="324"/>
</MIBObject>
```

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

```
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"
</mg/mib/ipNetToMediaTable>;rt="core.mg.mib";oid="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 may not.

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.

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

Setting up parameter values and establishing relations between devices during commissioning of a managed network is needed for the TRAP function. Draft [I-D.ietf-core-interfaces] describes the binding of end-points to end-points on remote devices. This is just a table that contains the destination addresses of the MIB variables. A list of objects describing different aspects of commissioning comprise:

- Binding table as described in [I-D.ietf-core-interfaces], schema presented in Appendix B.1.
- Notification sources as referred to in [RFC3416], schema presented in Appendix B.1.
- Names of files containing the schemas to be expected, schema presented in Appendix B.2.

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.

The object of type "Schema-files" contains a sequence of schema files describing the data structure transportable in CoMI messages.

8. Security Considerations

TODO: follows CoAP security provisioning.

9. IANA Considerations

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

Content types to be registered:

- application/comi+xml
- application/comi+exi
- 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. The draft has benefited from comments by Dee Denteneer, Esko Dijk, and Michael van Hartskamp.

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

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.

[I-D.bormann-cbor] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-bormann-cbor-09 (work in progress), September 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-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-10 (work in progress), September 2013.

[I-D.ietf-json-rfc4627bis] Bray, T., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-06 (work in progress), October 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.

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

[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.

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

[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-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-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-16 (work in progress), October 2013.

[I-D.ietf-core-interfaces]
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-00 (work in progress), June 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.

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

[EXI] , "Efficient XML Interchange", Web http://www.w3.org/xml/exi, .

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

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

[EXI-primer]
Peintner, D. and S. Pericas-Geertsen, "EXI primer", Web http://www.w3.org/TR/exi-primer, december 2009.

[EXI-measurement]
White, G., KangaSharju, J., Williams, S., and D. Brutzman, "Efficient XML Interchange Measurements Note", Web http://www.w3.org/TR/2007/WD-exi-measurements-20070725, July 2007.

[JSON-XML]
Nurseitov, N., Paulson, M., Reynolds, R., and C. Inzurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study", Web http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf, 2009.

Appendix A. XML Schema for MIB

This appendix describes the XML schema that defines the payload contents for MIB requests via the CoMI Function Set. It is assumed that MIB variables are referred by descriptor or by OID.

TODO: The schema needs to be updated to define basic types and notifications. Access may be more sophisticated than described here.

```
<xsd:simpleType name="bits">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="bit0"/>
  <xsd:enumeration value="bit1"/>
  <xsd:enumeration value="bit2"/>
  <xsd:enumeration value="bit3"/>
  <xsd:enumeration value="bit4"/>

  <xsd:enumeration value="bitn"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="exception">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="noSuchObject"/>
     <xsd:enumeration value="noSuchInstance"/>
     <xsd:enumeration value="endOfMibView"/>
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="error-status">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="noError"/>
     <xsd:enumeration  value="tooBig"/>
     <xsd:enumeration  value="noSuchName"/>
     <xsd:enumeration  value="badValue"/>
     <xsd:enumeration  value="readOnly"/>
     <xsd:enumeration  value="genErr"/>
     <xsd:enumeration  value="noAccess"/>
     <xsd:enumeration  value="wrongType"/>
     <xsd:enumeration  value="wrongLength"/>
     <xsd:enumeration  value="wrongEncoding"/>
     <xsd:enumeration  value="wrongValue"/>
     <xsd:enumeration  value="noCreation"/>
     <xsd:enumeration  value="inconsistentValue"/>
     <xsd:enumeration  value="resourceUnavailable"/>
     <xsd:enumeration  value="commitFailed"/>
     <xsd:enumeration  value="undoFailed"/>
     <xsd:enumeration  value="authorizationError"/>
```

```
        <xsd:enumeration  value="notWritable"/>
        <xsd:enumeration  value="inconsistentName"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:complexType name="MIBscalar">
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:choice>
       <xsd:attribute name="value" type="xsd:string"/>
       <xsd:attribute name="value" type="xsd:integer"/>
      </xsd:choice>
   </xsd:complexType>

   <xsd:complexType name="MIBvalue">
      <xsd:choice>
       <xsd:attribute name="value" type="xsd:string"/>
       <xsd:attribute name="value" type="xsd:integer"/>
      </xsd:choice>
   </xsd:complexType>


   <xsd:complexType name ="oid">
      <xsd:sequence>
       <xsd:element name="ident" type="xsd:integer"
        minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name ="MIBidentifier">
      <xsd:choice>
       <xsd:element name="descriptor" type="xsd:string"/>
        <xsd:element name="OID" type="oid"/>
      </xsd:choice>
   </xsd:complexType>


   <xsd:complexType name="row">
      <xsd:sequence>
        <xsd:element name="entry" type="MIBscalar"
           minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>


   <xsd:complexType name="MIBTable">
      <xsd:element name="MIBname" type="MIBidentifier"/>
      <xsd:sequence>
         <xsd:element name="Row" type="row"
```

```
           minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:complexType name="MIBObject">
       <xsd:element name="MIBname" type="MIBidentifier"/>
       <xsd:element name="value" type="MIBvalue"/>
   </xsd:complexType>
```

Appendix B. XML Schema for CoMI support

This appendix describes the XML schema that defines the payload contents for requests via the CoMI Function Set to the CoMI objects for multicast group, binding table, and SNMP notifications. For the SNMP notifications the Binding Method table specification of [I-D.ietf-core-interfaces] has been extended with "notify".

B.1. Schema for CoAP binding

Binding table contains several simple Bindings, composed of timing parameters and Function signature.

```
   <xs:complexType name="CoAPmethod">
       <xs:restriction base="xs:string">
          <xs:enumeration value="GET"/>
          <xs:enumeration value="PUT"/>
          <xs:enumeration value="POST"/>
       </xs:restriction>
   </xs:complexType>

   <xs:complexType name="bindingMethod">
       <xs:restriction base="xs:string">
     <xs:enumeration value="poll"/>
     <xs:enumeration value="obs"/>
     <xs:enumeration value="push"/>
     <xs:enumeration value="notify"/>
       </xs:restriction>
   </xs:complexType>


   <xs:complexType name="invocation">
     <xs:element name="hostname" type="xs:string"/>
     <xs:element name="pathname" type="xs:string"/>
     <xs:element name="IPaddress" type="xs:string"/>
     <xs:element name="bindingMethod" type="bindingMethod"/>
     <xs:element name="CoAPmethod" type="CoAPmethod"/>
   </xs:complexType>
```

```
<xs:complexType name="simpleBinding">
     <xs:element name="method" type="invocation"/>
     <xs:element name="minPeriod" type="xs:integer"/>
     <xs:element name="maxPeriod" type="xs:integer"/>
     <xs:element name="changeStep" type="xs:integer"/>
</xs:complexType>

<xs:complexType name="binding Table">
  <xs:sequence>
      <xs:element name="simpleBinding" type="simpleBinding"
          minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

B.2. Valid Schemas

File names are stored in Schema

```
<xs:complexType name="Schema-files">
<xs:sequence>
      <xs:element name="Schema" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
```

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