Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 26, 2019

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
I. Petrov, Ed.
Acklio

March 25, 2019

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-08

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

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 https://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 September 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
2. Terminology and Notation 3
2.1. YANG Schema Item iDentifier (SID) 5
2.2. CBOR diagnostic notation 5
3. Properties of the CBOR Encoding 6
4. Encoding of YANG Schema Node Instances 7
4.1. The 'leaf' 7
4.2. The 'container' and other collections 8
4.2.1. SIDs as keys 8
4.2.2. Member names as keys 9
4.3. The 'leaf-list' 11
4.4. The 'list' and 'list' instance(s) 11
4.4.1. SIDs as keys 12
4.4.2. Member names as keys 14
4.5. The 'anydata' 15
4.6. The 'anyxml' 17
5. Encoding of YANG data templates 17
5.1. SIDs as keys 18
5.2. Member names as keys 19
6. Representing YANG Data Types in CBOR 20
6.1. The unsigned integer Types 20
6.2. The integer Types 21
6.3. The 'decimal64' Type 21
6.4. The 'string' Type 22
6.5. The 'boolean' Type 22
6.6. The 'enumeration' Type 22
6.7. The 'bits' Type 23
6.8. The 'binary' Type 24
6.9. The 'leafref' Type 24
6.10. The 'identityref' Type 25
6.10.1. SIDs as identityref 25
6.10.2. Name as identityref 26
6.11. The 'empty' Type 26
6.12. The 'union' Type 27
6.13. The 'instance-identifier' Type 28
6.13.1. SIDs as instance-identifier 28
6.13.2. Names as instance-identifier 31
7. Security Considerations 32

8. IANA Considerations 32
8.1. Tags Registry 32
9. Acknowledgments 33
10. References 33
10.1. Normative References 33
10.2. Informative References 34
Authors' Addresses 34

1. Introduction

The specification of the YANG 1.1 data modelling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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].

The following terms are defined in [RFC7950]:

- action
- anydata
- anyxml
- data node
- data tree
- datastore
- feature

- identity
- module
- notification
- RPC
- schema node
- schema tree
- submodule

The following terms are defined in [RFC7951]:

- member name
- name of an identity
- namespace-qualified

The following terms are defined in [RFC8040]:

- yang-data (YANG extension)
- YANG data template

This specification also makes use of the following terminology:

- child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- parent: The collection in which a schema node is defined.
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

2.1. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier (SID), is encoded using an unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize its size, in certain positions, SIDs are represented using a (signed) delta from a reference SID and the current SID. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness is outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is under development as [I-D.ietf-core-sid].

2.2. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7B
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7A
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'F15C'	42 f15C
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187B 021901C8
Boolean	7/20	false	false	F4
	7/21	true	true	F5
Null	7/22	null	null	F6
Not assigned	7/23	undefined	undefined	F7

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- Any text within and including a pair of slashes is considered a comment.

- Deltas are visualized as numbers preceded by a '+' or '-' sign. The use of the '+' sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using

a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 2.1 and member names as defined in [RFC7951]. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of anyxml schema nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

Unless specified otherwise by the protocol or mechanism implementing this specification, the infinite lengths encoding as defined in [RFC7049] section 2.2 SHALL be supported by CBOR decoders.

Data nodes implemented using a CBOR array, map, byte string, and text string can be instantiated but empty. In this case, they are encoded with a length of zero.

Application payloads carrying a value serialized using the rules defined by this specification (e.g. CoAP Content-Format) SHOULD include the identifier (e.g. SID, namespace-qualified member name, instance-identifier) of this value. When SIDs are used as identifiers, the reference SID SHALL be included in the payload to allow stateless conversion of delta values to SIDs. Formats of these application payloads are not defined by the current specification and are not shown in the examples.

4. Encoding of YANG Schema Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The 'leaf'

A 'leaf' MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

4.2. The 'container' and other collections

Collections such as containers, list instances, notification contents, rpc inputs, rpc outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a schema node identifier, each value is set to the value of this schema node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 2.1 and member names as defined in [RFC7951].

The following examples shows the encoding of a 'system-state' container instance using SIDs or member names.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

4.2.1. SIDs as keys

CBOR map keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual delta value. Delta values are computed as follows:

- In the case of a 'container', deltas are equal to the SID of the current schema node minus the SID of the parent 'container'.

- In the case of an 'rpc input' or 'rcp output', deltas are equal to the SID of the current schema node minus the SID of the 'rpc'.

- In the case of an 'action input' or 'action output', deltas are equal to the SID of the current schema node minus the SID of the 'action'.

CBOR diagnostic notation:

```
{                                    / system-state (SID 1720) /
  +1 : {                             / clock  (SID 1721) /
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1723)/
    +1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1722) /
  }
}
```

CBOR encoding:

```
A1                                    # map(1)
   01                                 # unsigned(1)
   A2                                 # map(2)
      02                              # unsigned(2)
      78 1A                           # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
      01                             # unsigned(1)
      78 1a                          # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.2.2. Member names as keys

CBOR map keys implemented using member names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified member name MUST be used each time the namespace of a schema node and its parent differ. In all other cases, the simple form of the member name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names.

Definition example from [RFC7317]:

```
   typedef date-and-time {
     type string {
       pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
                \d{2}:\d{2})';
     }
   }

   container system-state {

     container clock {
       leaf current-datetime {
         type date-and-time;
       }

       leaf boot-datetime {
         type date-and-time;
       }
     }
   }
```

CBOR diagnostic notation:

```
{
  "ietf-system:clock" : {
    "current-datetime" : "2015-10-02T14:47:24Z-05:00",
    "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   71                                   # text(17)
      696574662D73797374656D3A636C6F636B # "ietf-system:clock"
   A2                                   # map(2)
      70                                # text(16)
         63757272656E742D6461746574696D65 # "current-datetime"
      78 1A                             # text(26)
         323031352D31302D30325431343A34373A32345A2D30353A3030
      6D                                # text(13)
         626F6F742D6461746574696D65     # "boot-datetime"
      78 1A                             # text(26)
         323031352D30392D31355430393A31323A35385A2D30353A3030
```

4.3. The 'leaf-list'

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

The following example shows the encoding of the 'search' leaf-list instance containing two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

```
typedef domain-name {
  type string {
    length "1..253";
    pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9].)
             *([a-zA-Z0-9_]([a-zA-Z0-9\-_]){0,61})?[a-zA-Z0-9]\.?
             )|\.';
  }
}

leaf-list search {
  type domain-name;
  ordered-by user;
}
```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662E6F7267 68 696565652E6F7267

4.4. The 'list' and 'list' instance(s)

A list or a subset of a list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the encoding rules of a collection as defined in Section 4.2.

It is important to note that this encoding rule also apply to a single 'list' instance.

The following examples show the encoding of a 'server' list using SIDs or member names.

Definition example from [RFC7317]:

```
list server {
  key name;

  leaf name {
    type string;
  }
  choice transport {
    case udp {
      container udp {
        leaf address {
          type host;
          mandatory true;
        }
        leaf port {
          type port-number;
        }
      }
    }
  }
  leaf association-type {
    type enumeration {
      enum server;
      enum peer;
      enum pool;
    }
    default server;
  }
  leaf iburst {
    type boolean;
    default false;
  }
  leaf prefer {
    type boolean;
    default false;
  }
}
```

4.4.1. SIDs as keys

The encoding rules of each 'list' instance are defined in Section 4.2.1. Deltas of list members are equal to the SID of the current schema node minus the SID of the 'list'.

CBOR diagnostic notation:

```
[                                   / server (SID 1756) /
  {
    +3 : "NRC TIC server",          / name (SID 1759) /
    +5 : {                          / udp (SID 1761) /
      +1 : "tic.nrc.ca",            / address (SID 1762) /
      +2 : 123                      / port (SID 1763) /
    },
    +1 : 0,                         / association-type (SID 1757) /
    +2 : false,                     / iburst (SID 1758) /
    +4 : true                       / prefer (SID 1760) /
  },
  {
    +3 : "NRC TAC server",          / name (SID 1759) /
    +5 : {                          / udp (SID 1761) /
      +1 : "tac.nrc.ca"             / address (SID 1762) /
    }
  }
]
```

CBOR encoding:

```
82                                  # array(2)
   A5                               # map(5)
      03                            # unsigned(3)
      6E                            # text(14)
         4E524320544943207365726572 # "NRC TIC server"
      05                            # unsigned(5)
      A2                            # map(2)
         01                         # unsigned(1)
         6A                         # text(10)
            7469632E6E72632E6361    # "tic.nrc.ca"
         02                         # unsigned(2)
         18 7B                      # unsigned(123)
      01                            # unsigned(1)
      00                            # unsigned(0)
      02                            # unsigned(2)
      F4                            # primitive(20)
      04                            # unsigned(4)
      F5                            # primitive(21)
   A2                               # map(2)
      03                            # unsigned(3)
      6E                            # text(14)
         4E524320544143207365726572 # "NRC TAC server"
      05                            # unsigned(5)
      A1                            # map(1)
         01                         # unsigned(1)
         6A                         # text(10)
            7461632E6E72632E6361    # "tac.nrc.ca"
```

4.4.2. Member names as keys

The encoding rules of each 'list' instance are defined in Section 4.2.2.

CBOR diagnostic notation:

```
[
  {
    "ietf-system:name" : "NRC TIC server",
    "ietf-system:udp" : {
      "address" : "tic.nrc.ca",
      "port" : 123
    },
    "ietf-system:association-type" : 0,
    "ietf-system:iburst" : false,
    "ietf-system:prefer" : true
  },
  {
    "ietf-system:name" : "NRC TAC server",
    "ietf-system:udp" : {
      "address" : "tac.nrc.ca"
    }
  }
]
```

CBOR encoding:

```
82                                  # array(2)
   A5                               # map(5)
      70                            # text(16)
         696574662D73797374656D3A6E616D65 # "ietf-system:name"
      6E                            # text(14)
         4E5243205449432073657276657220   # "NRC TIC server"
      6F                            # text(15)
         696574662D73797374656D3A756470 # "ietf-system:udp"
      A2                            # map(2)
         67                         # text(7)
            61646472657373          # "address"
         6A                         # text(10)
            7469632E6E72632E6361    # "tic.nrc.ca"
         64                         # text(4)
            706F7274                # "port"
         18 7B                      # unsigned(123)
      78 1C                         # text(28)
         696574662D73797374656D3A6173736F63696174696F6E2D74797065
      00                            # unsigned(0)
      72                            # text(18)
         696574662D73797374656D3A696275727374 # "ietf-system:iburst"
      F4                            # primitive(20)
      72                            # text(18)
         696574662D73797374656D3A707265666572 # "ietf-system:prefer"
      F5                            # primitive(21)
   A2                               # map(2)
      70                            # text(16)
         696574662D73797374656D3A6E616D65 # "ietf-system:name"
      6E                            # text(14)
         4E5243205441432073657276657220   # "NRC TAC server"
      6F                            # text(15)
         696574662D73797374656D3A756470 # "ietf-system:udp"
      A1                            # map(1)
         67                         # text(7)
            61646472657373          # "address"
         6A                         # text(10)
            7461632E6E72632E6361    # "tac.nrc.ca"
```

4.5. The 'anydata'

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- CBOR map keys of any inner schema nodes MUST be set to valid deltas or member names.

- The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).

- CBOR map values MUST follow the encoding rules of one of the datatypes listed in Section 4.

The following example shows a possible use of an anydata. In this example, an anydata is used to define a schema node containing a notification event, this schema node can be part of a YANG list to create an event logger.

Definition example:

```
module event-log {
  ...
  anydata event;                    # SID 60123
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault {  # SID 60200
    leaf port-name {                 # SID 60201
      type string;
    }
    leaf port-fault {                # SID 60202
      type string;
    }
  }
}
```

CBOR diagnostic notation:

```
{                          / event (SID=60123) /
  +78 : "0/4/21",          / port-name (SID=60201) /
  +79 : "Open pin 2"       / port-fault (SID=60202) /
}
```

CBOR encoding:

```
A2                          # map(2)
   18 4E                    # unsigned(78)
   66                       # text(6)
      302F342F3231          # "0/4/21"
   18 4F                    # unsigned(79)
   6A                       # text(10)
      4F70656E2070696E2032 # "Open pin 2"
```

4.6. The 'anyxml'

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value MAY contain CBOR data items tagged with one of the tag listed in Section 8.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance consisting of a CBOR array containing the CBOR simple values 'true', 'null' and 'true'.

Definition example from [RFC7951]:

```
anyxml bar;
```

CBOR diagnostic notation: [true, null, true]

CBOR encoding: 83 f5 f6 f5

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the 'yang-data' extension as described by RFC 8040.

YANG data templates SHOULD be encoded using the encoding rules of a collection as defined in Section 4.2.

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

Definition example from [I-D.ietf-core-comi]:

```
import ietf-restconf {
  prefix rc;
}

rc:yang-data yang-errors {
  container error {
    leaf error-tag {
      type identityref {
        base error-tag;
      }
    }
    leaf error-app-tag {
      type identityref {
        base error-app-tag;
      }
    }
    leaf error-data-node {
      type instance-identifier;
    }
    leaf error-message {
      type string;
    }
  }
}
```

5.1. SIDs as keys

This example shows a serialization example of the yang-errors template using SIDs as CBOR map key. The reference SID of a YANG data template is zero, this imply that the CBOR map keys of the top level members of the template are set to SIDs.

CBOR diagnostic notation:

```
{
  1024 : {                     / error  (SID 1024) /
    +4 : 1011,                 / error-tag (SID 1028) /
                               / = invalid-value (SID 1011) /
    +1 : 1018,                 / error-app-tag (SID 1025) /
                               / = not-in-range (SID 1018) /
    +2 : 1740,                 / error-data-node (SID 1026) /
                               / = timezone-utc-offset (SID 1740) /
    +3 : "Maximum exceeded"    / error-message (SID 1027) /
      }
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 0400                              # unsigned(1024)
   A4                                   # map(4)
      04                                # unsigned(4)
      19 03F3                           # unsigned(1011)
      01                                # unsigned(1)
      19 03FA                           # unsigned(1018)
      02                                # unsigned(2)
      19 06CC                           # unsigned(1740)
      03                                # unsigned(3)
      70                                # text(16)
         4D6178696D756D206578636565646564
```

5.2. Member names as keys

This example shows a serialization example of the yang-errors template using member names as CBOR map key.

CBOR diagnostic notation:

```
{
  "ietf-comi:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "Maximum exceeded"
  }
}
```

CBOR encoding:

```
A1                                  # map(1)
   6F                               # text(15)
      696574662D636F6D693A6572726F72 # "ietf-comi:error"
   A4                               # map(4)
      69                            # text(9)
         6572726F722D746167         # "error-tag"
      6D                            # text(13)
         696E76616C69642D76616C7565 # "invalid-value"
      6D                            # text(13)
         6572726F722D6170702D746167 # "error-app-tag"
      6C                            # text(12)
         6E6F742D696E2D72616E6765   # "not-in-range"
      6F                            # text(15)
         6572726F722D646174612D6E6F6465 # "error-data-node"
      73                            # text(19)
         74696D657A6F6E652D7574632D6F6666736574 # "timezone-utc-offset"
      6D                            # text(13)
         6572726F722D6D657373616765 # "error-message"
      70                            # text(16)
         4D6178696D756D20657836565646564
```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list schema node depends on the built-in type of that schema node. The following sub-section defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a schema node instance of the discussed built-in type.

6.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

The following example shows the encoding of a 'mtu' leaf instance set to 1280 bytes.

Definition example from [RFC7277]:

```
leaf mtu {
  type uint16 {
    range "68..max";
  }
}
```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

```
leaf timezone-utc-offset {
  type int16 {
    range "-1500 .. 1500";
  }
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012B

6.3. The 'decimal64' Type

Leafs of type decimal64 MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

```
leaf my-decimal {
  type decimal64 {
    fraction-digits 2;
    range "1 .. 3.14 | 10 | 20..max";
  }
}
```

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: C4 82 21 19 0101

6.4. The 'string' Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

The following example shows the encoding of a 'name' leaf instance set to "eth0".

Definition example from [RFC7223]:

```
leaf name {
  type string;
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

Leafs of type boolean MUST be encoded using a CBOR simple value 'true' (major type 7, additional information 21) or 'false' (major type 7, additional information 20).

The following example shows the encoding of an 'enabled' leaf instance set to 'true'.

Definition example from [RFC7317]:

```
leaf enabled {
  type boolean;
}
```

CBOR diagnostic notation: true

CBOR encoding: F5

6.6. The 'enumeration' Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an 'oper-status' leaf instance set to 'testing'.

Definition example from [RFC7317]:

```
leaf oper-status {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
    enum unknown { value 4; }
    enum dormant { value 5; }
    enum not-present { value 6; }
    enum lower-layer-down { value 7; }
  }
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

6.7. The 'bits' Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of a 'mybits' leaf instance with the 'disable-nagle' and '10-Mb-only' flags set.

Definition example from [RFC7950]:

```
leaf mybits {
  type bits {
    bit disable-nagle {
      position 0;
    }
    bit auto-sense-speed {
      position 1;
    }
    bit 10-Mb-only {
      position 2;
    }
  }
}
```

CBOR diagnostic notation: h'05'

CBOR encoding: 41 05

6.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

```
leaf aes128-key {
  type binary {
    length 16;
  }
}
```

CBOR diagnostic notation: h'1F1CE6A3F42660D888D92A4D8030476E'

CBOR encoding: 50 1F1CE6A3F42660D888D92A4D8030476E

6.9. The 'leafref' Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the 'path' YANG statement.

The following example shows the encoding of an 'interface-state-ref' leaf instance set to "eth1".

Definition example from [RFC7223]:

```
typedef interface-state-ref {
  type leafref {
    path "/interfaces-state/interface/name";
  }
}

container interfaces-state {
  list interface {
    key "name";
    leaf name {
      type string;
    }
    leaf-list higher-layer-if {
      type interface-state-ref;
    }
  }
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding identityref, a YANG Schema Item iDentifier (SID) as defined in Section 2.1 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as identityref

When schema nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as identityref.)

The following example shows the encoding of a 'type' leaf instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1880).

Definition example from [RFC7317]:

```
identity interface-type {
}

identity iana-interface-type {
  base interface-type;
}

identity ethernetCsmacd {
  base iana-interface-type;
}

leaf type {
  type identityref {
    base interface-type;
  }
}
```

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

6.10.2. Name as identityref

Alternatively, an identityref MAY be encoded using a name as defined in [RFC7951] section 6.8. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in different module than the leaf node containing the identityref value, the namespace-qualified form MUST be used. Otherwise, both the simple and namespace-qualified forms are permitted. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616E612D69662D747970653A65746865726E657443736D616364

6.11. The 'empty' Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC7277]:

```
leaf is-router {
  type empty;
}
```

CBOR diagnostic notation: null

CBOR encoding: F6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are prefixed by CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 8.1 for the assigned value of these CBOR tags.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

```
typedef ipv4-address {
  type string {
  pattern '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}
           ([0-9][1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(%[\p{N}
           \p{L}]+)?';
  }
}

typedef ipv6-address {
  type string {
    pattern '((:|[0-9a-fA-F]{0,4}):)([0-9a-fA-F]{0,4}:){0,5}((([0-9a
             -fA-F]{0,4}:)?(:|[0-9a-fA-F]{0,4}))|(((25[0-5]|2[0-4][0
             -9]|[01]?[0-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0
             -9]?[0-9])))(%[\p{N}\p{L}]+)?';
    pattern '(([^:]+:){6}(([^:]+:[^:]+)|(.*\..*)))|((([^:]+:)*[^:]+)
             ?::(([^:]+:)*[^:]+)?)(%.+)?';
  }
}

typedef ip-address {
  type union {
    type ipv4-address;
    type ipv6-address;
  }
}

leaf address {
  type inet:ip-address;
}
```

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313A6462383A6130623A313266303A3A31

6.13. The 'instance-identifier' Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 2.1 and one based on names as defined in [RFC7951] section 6.11.

6.13.1. SIDs as instance-identifier

SIDs uniquely identify a schema node. In the case of a single instance schema node, i.e. a schema node defined at the root of a YANG module or submodule or schema nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a schema node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance schema nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.

Schema nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.

- The following entries MUST contain the value of each key required to identify the instance of the targeted schema node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

Examples within this section assume the definition of a schema node of type 'instance-identifier':

Definition example from [RFC7950]:

```
container system {
  ...
  leaf reporting-entity {
    type instance-identifier;
  }

leaf contact { type string; }

leaf hostname { type inet:domain-name; } } ~~~~
```

First example:

The following example shows the encoding of the 'reporting-entity' value referencing data node instance "/system/contact" (SID 1741).

Definition example from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

  leaf hostname {
    type inet:domain-name;
  }
}
```

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

The following example shows the encoding of the 'reporting-entity' value referencing list instance "/system/authentication/user/authorized-key/key-data" (SID 1734) for user name "bob" and authorized-key "admin".

Definition example from [RFC7317]:

```
list user {
  key name;

  leaf name {
    type string;
  }
  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key name;

    leaf name {
      type string;
    }
    leaf algorithm {
      type string;
    }
    leaf key-data {
      type binary;
  }
}
```

CBOR diagnostic notation: [1734, "bob", "admin"]

CBOR encoding:

```
83               # array(3)
   19 06C6       # unsigned(1734)
   63            # text(3)
      626F62     # "bob"
   65            # text(5)
      61646D696E # "admin"
```

Third example:

The following example shows the encoding of the 'reporting-entity' value referencing the list instance "/system/authentication/user" (SID 1730) corresponding to user name "jack".

CBOR diagnostic notation: [1730, "jack"]

CBOR encoding:

```
82             # array(2)
   19 06C2     # unsigned(1730)
   64          # text(4)
      6A61636B # "jack"
```

6.13.2. Names as instance-identifier

The use of names as instance-identifier is defined in [RFC7951] section 6.11. The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

CBOR diagnostic notation: "/ietf-system:system/contact"

CBOR encoding:

```
78 1c 2F696574662D73797374656D3A73797374656D2F636F6E74616374
```

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']/authorized-key
[name='admin']/key-data"

CBOR encoding:

```
78 59
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A
   65642D6B65790D0A5B6E616D653D2761646D696E275D2F6B65792D64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']"

CBOR encoding:

```
78 33
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D27626F62275D
```

7. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

8. IANA Considerations

8.1. Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the Tags Registry as defined in section 7.2 of [RFC7049].

Tag	Data Item	Semantics	Reference
xx	bits	YANG bits datatype	RFC XXXX
xx	enumeration	YANG enumeration datatype	RFC XXXX
xx	identityref	YANG identityref datatype	RFC XXXX
xx	instance-identifier	YANG instance-identifier datatype	RFC XXXX

// RFC Ed.: update Tag values using allocated tags and remove this note // RFC Ed.: replace XXXX with RFC number and remove this note

9. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

10.2. Informative References

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-04 (work in progress), November 2018.

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-05 (work in progress), December 2018.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <https://www.rfc-editor.org/info/rfc7223>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <https://www.rfc-editor.org/info/rfc7277>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Authors' Addresses

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

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

Alexander Pelov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn, Vorarlberg 6850
Austria

Phone: +43664808926169
Email: abhinav.somaraju@tridonic.com

Randy Turner
Landis+Gyr
30000 Mill Creek Ave
Suite 100
Alpharetta, GA 30022
US

Phone: ++16782581292
Email: randy.turner@landisgyr.com
URI: http://www.landisgyr.com/

Ana Minaburo
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ana@ackl.io

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io