Stream: Internet Engineering Task Force (IETF)
RFC: 9254
Category: Standards Track
Published: July 2022
ISSN: 2070-1721
Authors: M. Veillette, Ed. *Trilliant Networks Inc.*
I. Petrov, Ed. *Google Switzerland GmbH*
A. Pelov *Acklio*
C. Bormann *Universität Bremen TZI*
M. Richardson *Sandelman Software Works*

RFC 9254
Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR)

Abstract

YANG (RFC 7950) is a data modeling language used to model configuration data, state data, parameters and results of Remote Procedure Call (RPC) operations or actions, and notifications.

This document defines encoding rules for YANG in the Concise Binary Object Representation (CBOR) (RFC 8949).

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9254.

Copyright Notice

Table of Contents

1. Introduction

The specification of the YANG 1.1 data modeling 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.

An additional set of encoding rules has been defined in [RFC7951] based on "The JavaScript Object Notation (JSON) Data Interchange Format" [RFC8259].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC8949], collectively called "YANG-CBOR". 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**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

SID values (and the SID deltas computed from them) shown in the examples are example values; these examples do not allocate the SIDs shown for specific items in the modules.

The following terms are defined in [RFC7950]:

- action
- anydata
- anyxml
- data node
- data tree
- datastore
- feature
- identity
- module
- notification
- RPC
- schema node

- submodule

The following term is defined in [RFC8040]:

- yang-data extension

The following term is defined in [RFC8791]:

- YANG data structure

This specification also makes use of the following terminology:

YANG Schema Item iDentifier (or "YANG SID" or simply "SID"):
: 63-bit unsigned integer used to identify different YANG items.

delta:
: Difference between the current YANG SID and a reference YANG SID. A reference YANG SID is defined for each context for which deltas are used.

absolute SID:
: A YANG SID that is not encoded as a delta. This is usually called out explicitly only in positions where normally a delta would be found.

representation tree:
: A YANG data tree, possibly enclosed by a representation of a schema node, such as a YANG data structure, a notification, an RPC, or an action.

representation node:
: A node in a representation tree, i.e., a data tree node, or a representation of a schema node, such as a YANG data structure, a notification, an RPC, or an action.

item:
: A schema node, an identity, a module, or a feature defined using the YANG modeling language.

list entry:
: The data associated with a single entry of a list (see Section 7.8 of [RFC7950]).

container-like instance:
: An instance of a container, a YANG data structure, notification contents, RPC input, RPC output, action input, or action output (Section 4.2); a list entry in a list (Section 4.4); or an anydata node (Section 4.5).

parent (of a representation node):
: The schema node of the closest enclosing representation node in which a given representation node is defined.

3. Properties of the CBOR Encoding

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

A YANG data tree can be enclosed by a representation of a schema node, such as a YANG data structure, a notification, an RPC, or an action; this is called a representation tree. The data tree nodes and the enclosing schema node representation, if any, are collectively called the representation nodes.

A representation node, such as a container, list entry, YANG data structure, notification, RPC input, RPC output, action input, action output, or anydata node, is serialized using a CBOR map in which each schema node defined within is encoded using a key and a value. This specification supports two types of CBOR keys: YANG Schema Item iDentifier (YANG SID), as defined in Section 3.2, and names, as defined in Section 3.3. Each of these key types is encoded using a specific CBOR type that allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type or allow the generator to choose freely per key.

In order to minimize the size of the encoded data, the mapping avoids any unnecessary meta-information beyond that directly provided by the CBOR basic generic data model (Section 2 of [RFC8949]). For instance, CBOR tags are used solely in the case of an absolute SID, anyxml data nodes, or the union datatype to explicitly distinguish 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 indefinite length encoding, as defined in Section 3.2 of [RFC8949], **SHALL** be supported by the CBOR decoders employed with YANG-CBOR. (This enables an implementation to begin emitting an array or map before the number of entries in that structure is known, possibly also avoiding excessive locking or race conditions. On the other hand, it deprives the receiver of the encoded data from advance announcement about some size information, so a generator should choose indefinite length encoding only when these benefits do accrue.)

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

When representation nodes are serialized using the rules defined by this specification as part of an application payload, the payload **SHOULD** include information that would allow each node to be identified in a stateless way, for instance, the SID number associated with the node, the SID delta from another SID in the application payload, the namespace-qualified name, or the instance-identifier.

Examples in Section 4 include a root CBOR map with a single entry having a key set to either a namespace-qualified name or a SID. This root CBOR map is provided only as a typical usage example and is not part of the present encoding rules. Only the value within this CBOR map is compulsory.

3.1. CBOR Diagnostic Notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation, as defined in Section 8 of [RFC8949]. 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

Note: CBOR binary contents shown in this specification are annotated with comments. These comments are delimited by slashes ("/"), as defined in Appendix G.6 of [RFC8610].

3.2. YANG Schema Item iDentifier

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both the Network Configuration Protocol (NETCONF) [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using text strings. 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", is an unsigned integer limited to 63 bits of range (i.e., 0..9223372036854775807 or 0..0x7fffffffffffffff). The following items are identified using YANG SIDs (often shortened to SIDs):

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

> Note that any structuring of modules into submodules is transparent to YANG-CBOR: SIDs are not allocated for the names of submodules, and any items within a submodule are effectively allocated SIDs as part of processing the module that includes them.

To minimize their size, SIDs used as keys in CBOR maps are encoded using deltas, i.e., signed (negative or unsigned) integers that are added to the reference SID applying to the map. The reference SID of an outermost map is zero, unless a different reference SID is unambiguously conferred from the environment in which the outermost map is used. The reference SID of a map that is most directly embedded in a map entry with a name-based key is zero. For all other maps, the reference SID is the SID computed for the map entry it is most directly embedded in. (The embedding may be indirect if an array intervenes, e.g., in a YANG list.) Where absolute SIDs are desired in map key positions (where a bare integer implies a delta), they need to be identified as absolute SID values by using CBOR tag number 47 (as defined in Section 4.2.1).

Thus, conversion from SIDs to deltas and back to SIDs is a stateless process solely based on the data serialized or deserialized combined with, potentially, an outermost reference SID unambiguously conferred by the environment.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness are 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. A related document, i.e., [CORE-SID], is intended to serve as the definitive way to assign SID values for YANG modules managed by the IETF and recommends itself for YANG modules managed by non-IETF entities, as well. The present specification has been designed to allow different methods of assignment to be used within separate domains.

To provide implementations with a way to internally indicate the absence of a SID, the SID value 0 is reserved and will not be allocated; it is not used in interchange.

3.3. Name

This specification also supports the encoding of YANG item identifiers as text strings, similar to those used by the JSON encoding of data modeled with YANG [RFC7951]. This approach can be used to avoid the management overhead associated with SID allocation. The main drawback is the significant increase in size of the encoded data.

YANG item identifiers implemented using names **MUST** be in one of the following forms:

- simple -- the identifier of the YANG item (i.e., schema node or identity).
- namespace-qualified -- the identifier of the YANG item is prefixed with the name of the module in which this item is defined, separated by the colon character (":").

The name of a module determines the namespace of all YANG items defined in that module. If an item is defined in a submodule, then the namespace-qualified name uses the name of the main module to which the submodule belongs.

ABNF syntax [RFC5234] of a name is shown in Figure 1, where the production for "identifier" is defined in Section 14 of [RFC7950].

```
name = [identifier ":"] identifier
```

Figure 1: ABNF Production for a Simple or Namespace-Qualified Name

A namespace-qualified name **MUST** be used for all members of a top-level CBOR map and then also whenever the namespaces of the representation node and its parent node are different. In all other cases, the simple form of the name **MUST** be used.

Definition example:

```
module example-foomod {
  container top {
    leaf foo {
      type uint8;
    }
  }
}

module example-barmod {
  import example-foomod {
    prefix "foomod";
  }
  augment "/foomod:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

A valid CBOR encoding of the 'top' container is as follows.

CBOR diagnostic notation:

```
{
  "example-foomod:top": {
    "foo": 54,
    "example-barmod:bar": true
  }
}
```

Both the 'top' container and the 'bar' leaf defined in a different YANG module as its parent container are encoded as namespace-qualified names. The 'foo' leaf defined in the same YANG module as its parent container is encoded as a simple name.

4. Encoding of Representation Nodes

Representation nodes defined using the YANG modeling language are encoded using CBOR [RFC8949], based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC8949].

4.1. The 'leaf'

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

The following examples show the encoding of a 'hostname' leaf using a SID or a name.

Definition example adapted from [RFC6991] and [RFC7317]:

```
typedef domain-name {
  type string {
    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]\.?)'
    + '|\.';
    length "1..253";
  }
}

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

4.1.1. Using SIDs in Keys

As with all examples below, the delta in the outermost map assumes a reference YANG SID (current schema node) of 0.

CBOR diagnostic notation:

```
{
  1752 : "myhost.example.com"      / hostname (SID 1752) /
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 06D8                              # unsigned(1752)
   72                                   # text(18)
      6D79686F73742E6578616D706C652E636F6D # "myhost.example.com"
```

4.1.2. Using Names in Keys

CBOR diagnostic notation:

```
{
  "ietf-system:hostname" : "myhost.example.com"
}
```

CBOR encoding:

```
A1                                      # map(1)
   74                                   # text(20)
      696574662D73797374656D3A686F73746E616D65
   72                                   # text(18)
      6D79686F73742E6578616D706C652E636F6D
```

4.2. The 'container' and Other Nodes from the Data Tree

Instances of containers, YANG data structures, notification contents, RPC inputs, RPC outputs, action inputs, and action outputs **MUST** be encoded using a CBOR map data item (major type 5). The same encoding is also used for the list entries in a list (Section 4.4) and for anydata nodes (Section 4.5). Collectively, we speak of these instances as "container-like instances".

A map consists of pairs of data items, with each pair consisting of a key and a value. Each key within the CBOR map is set to a schema node identifier, and each value is set to the value of this representation node according to the instance datatype.

This specification supports two types of CBOR map keys: SID, as defined in Section 3.2, and names, as defined in Section 3.3.

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

Definition example adapted from [RFC6991] and [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. Using SIDs in Keys

In the context of containers and other nodes from the data tree, CBOR map keys within inner CBOR maps can be encoded using deltas (bare integers) or absolute SIDs (tagged with tag number 47).

Delta values are computed as follows:

- In the case of a 'container', deltas are equal to the SID of the current representation node minus the SID of the parent 'container'.
- In the case of a 'list', deltas are equal to the SID of the current representation node minus the SID of the parent 'list'.
- In the case of an 'RPC input' or 'RPC output', deltas are equal to the SID of the current representation 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 representation node minus the SID of the 'action'.
- In the case of a 'notification content', deltas are equal to the SID of the current representation node minus the SID of the 'notification'.

CBOR diagnostic notation:

```
{
  1720 : {                              / 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)
   19 06B8                              # unsigned(1720)
   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
```

Figure 2: System State Clock Encoding

4.2.2. Using Names in Keys

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

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

CBOR diagnostic notation:

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

CBOR encoding:

```
A1                                      # map(1)
   78 18                                # text(24)
      696574662D73797374656D3A73797374656D2D7374617465
   A1                                   # map(1)
      65                                # text(5)
         636C6F636B                     # "clock"
      A2                                # map(2)
         70                             # text(16)
            63757272656E742D6461746574696D65
         78 1A                          # text(26)
            323031352D31302D30325431343A34373A32345A2D30353A3030
         6D                             # text(13)
            626F6F742D6461746574696D65
         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 representation node instance containing two entries: "ietf.org" and "ieee.org".

Definition example adapted from [RFC6991] and [RFC7317]:

```
typedef domain-name {
  type string {
    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]\.?)'
    + '|\.';
    length "1..253";
  }
}

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

4.3.1. Using SIDs in Keys

CBOR diagnostic notation:

```
{
  1746 : [ "ietf.org", "ieee.org" ]     / search (SID 1746) /
}
```

CBOR encoding:

```
A1                      # map(1)
   19 06D2              # unsigned(1746)
   82                   # array(2)
      68                # text(8)
         696574662E6F7267 # "ietf.org"
      68                # text(8)
         696565652E6F7267 # "ieee.org"
```

4.3.2. Using Names in Keys

CBOR diagnostic notation:

```
{
  "ietf-system:search" : [ "ietf.org", "ieee.org" ]
}
```

CBOR encoding:

```
A1                                      # map(1)
   72                                   # text(18)
      696574662D73797374656D3A736561726368 # "ietf-system:search"
   82                                   # array(2)
      68                                # text(8)
         696574662E6F7267               # "ietf.org"
      68                                # text(8)
         696565652E6F7267               # "ieee.org"
```

4.4. The 'list' and the 'list' Entries

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

It is important to note that this encoding rule also applies to a 'list' representation node instance that has a single entry.

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

Definition example adapted 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. Using SIDs in Keys

The encoding rules of each 'list' entry are defined in Section 4.2.1.

CBOR diagnostic notation:

```
{
  1756 : [                      / 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:

```
A1                                  # map(1)
   19 06DC                          # unsigned(1756)
   82                               # array(2)
      A5                            # map(5)
         03                         # unsigned(3)
         6E                         # text(14)
            4E5243205449432073657276657265 # "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)
            4E5243205441432073657276657265 # "NRC TAC server"
         05                         # unsigned(5)
         A1                         # map(1)
            01                      # unsigned(1)
            6A                      # text(10)
               7461632E6E72632E6361 # "tac.nrc.ca"
```

4.4.2. Using Names in Keys

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

CBOR diagnostic notation:

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

CBOR encoding:

```
A1                                      # map(1)
   72                                   # text(18)
      696574662D73797374656D3A736572766572
   82                                   # array(2)
      A5                                # map(5)
         64                             # text(4)
            6E616D65                    # "name"
         6E                             # text(14)
            4E5243205449432073657276657
         63                             # text(3)
            756470                      # "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)
         70                             # text(16)
            6173736F63696174696F6E2D74797065
         00                             # unsigned(0)
         66                             # text(6)
            696275727374                # "iburst"
         F4                             # primitive(20)
         66                             # text(6)
            707265666572                # "prefer"
         F5                             # primitive(21)
      A2                                # map(2)
         64                             # text(4)
            6E616D65                    # "name"
         6E                             # text(14)
            4E5243205441432073657276657
         63                             # text(3)
            756470                      # "udp"
         A1                             # map(1)
            67                          # text(7)
               61646472657373           # "address"
            6A                          # text(10)
               7461632E6E72632E6361     # "tac.nrc.ca"
```

4.5. The 'anydata'

An anydata node serves as a container for an arbitrary set of representation nodes that otherwise appear as normal YANG-modeled data. An anydata representation node instance is encoded using the same rules as a container, i.e., using a CBOR map data item (major type 5) based on the encoding rules of a container-like instance, as defined in Section 4.2.

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

Definition example:

```
module event-log {
  ...
  anydata last-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;
    }
  }
}
```

4.5.1. Using SIDs in Keys

CBOR diagnostic notation:

```
{
  60123 : {                   / last-event (SID 60123) /
    77 : {                    / example-port-fault (SID 60200) /
      1 : "0/4/21",           / port-name (SID 60201) /
      2 : "Open pin 2"        / port-fault (SID 60202) /
    }
  }
}
```

CBOR encoding:

```
A1                              # map(1)
   19 EADB                      # unsigned(60123)
   A1                           # map(1)
      18 4D                     # unsigned(77)
      A2                        # map(2)
         01                     # unsigned(1)
         66                     # text(6)
            302F342F3231        # "0/4/21"
         02                     # unsigned(2)
         6A                     # text(10)
            4F70656E2070696E2032 # "Open pin 2"
```

In some implementations, it might be simpler to use the absolute SID encoding (tag number 47) for the anydata root element. CBOR diagnostic notation:

```
{
  60123 : {                    / last-event (SID 60123) /
    47(60200) : {              / event-port-fault (SID 60200) /
      1 : "0/4/21",            / port-name (SID 60201) /
      2 : "Open pin 2"         / port-fault (SID 60202) /
    }
  }
}
```

4.5.2. Using Names in Keys

CBOR diagnostic notation:

```
{
  "event-log:last-event" : {
    "example-port:example-port-fault" : {
      "port-name" : "0/4/21",
      "port-fault" : "Open pin 2"
    }
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   74                                   # text(20)
      6576656E742D6C6F673A6C6173742D6576656E74
   A1                                   # map(1)
      78 1F                             # text(31)
         6578616D706C652D706F72743A
         6578616D706C652D706F72742D6661756C74
      A2                                # map(2)
         69                             # text(9)
            706F72742D6E616D65          # "port-name"
         66                             # text(6)
            302F342F3231                # "0/4/21"
         6A                             # text(10)
            706F72742D6661756C74        # "port-fault"
         6A                             # text(10)
            4F70656E2070696E2032        # "Open pin 2"
```

4.6. The 'anyxml'

An anyxml representation node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. (The "xml" in the name is a misnomer that only applied to YANG-XML [RFC7950].) An anyxml value **MAY** contain CBOR data items tagged with one of the tags listed in Section 9.3. The tags listed in Section 9.3 **SHALL** be supported.

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

Definition example adapted from [RFC7951]:

```
module bar-module {
  ...
  anyxml bar;      // SID 60000
}
```

4.6.1. Using SIDs in Keys

CBOR diagnostic notation:

```
{
  60000 : [true, null, true]   / bar (SID 60000) /
}
```

CBOR encoding:

```
A1         # map(1)
   19 EA60 # unsigned(60000)
   83      # array(3)
      F5   # primitive(21)
      F6   # primitive(22)
      F5   # primitive(21)
```

4.6.2. Using Names in Keys

CBOR diagnostic notation:

```
{
  "bar-module:bar" : [true, null, true]   / bar (SID 60000) /
}
```

CBOR encoding:

```
A1                                  # map(1)
   6E                               # text(14)
      6261722D6D6F64756C653A626172 # "bar-module:bar"
   83                               # array(3)
      F5                            # primitive(21)
      F6                            # primitive(22)
      F5                            # primitive(21)
```

5. Encoding of the 'yang-data' Extension

The yang-data extension [RFC8040] is used to define data structures in YANG that are not intended to be implemented as part of a datastore.

The yang-data extension will specify a container that **MUST** be encoded using the encoding rules of nodes of data trees, as defined in Section 4.2.

Just like YANG containers, the yang-data extension can be encoded using either SIDs or names.

Definition example adapted from Appendix A of [CORE-COMI]:

```
module ietf-coreconf {
  ...

  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. Using SIDs in Keys

The yang-data extensions encoded using SIDs are carried in a CBOR map containing a single item pair. The key of this item is set to the SID assigned to the yang-data extension container; the value is set to the CBOR encoding of this container, as defined in Section 4.2.

This example shows a serialization example of the yang-errors yang-data extension, as defined in [CORE-COMI], using SIDs, as defined in Section 3.2.

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 # "Maximum exceeded"
```

5.2. Using Names in Keys

The yang-data extensions encoded using names are carried in a CBOR map containing a single item pair. The key of this item is set to the namespace-qualified name of the yang-data extension container; the value is set to the CBOR encoding of this container, as defined in Section 4.2.

This example shows a serialization example of the yang-errors yang-data extension, as defined in [CORE-COMI], using names, as defined Section 3.3.

CBOR diagnostic notation:

```
{
  "ietf-coreconf: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)
   73                                       # text(19)
      696574662D636F7265636F6E663A6572726F72 # "ietf-coreconf: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)
         4D6178696D756D206578636565646564   # "Maximum exceeded"
```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list representation node depends on the built-in type of that representation node. The following subsection defines the CBOR encoding of each built-in type supported by YANG, as listed in Section 4.2.4 of [RFC7950]. Each subsection shows an example value assigned to a representation 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 an 'mtu' leaf representation node instance set to 1280 bytes.

Definition example adapted from [RFC8344]:

```
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 a CBOR unsigned integer (major type 0) or a CBOR negative integer (major type 1), depending on the actual value.

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

Definition example adapted 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 Section 3.4.4 of [RFC8949].

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

Definition example adapted 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 representation node instance set to "eth0".

Definition example adapted from [RFC8343]:

```
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 representation node instance set to 'true'.

Definition example adapted 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, or exceptionally as a tagged text string (see below). Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in Section 9.6.4.2 of [RFC7950].

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

Definition example adapted 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

Values of 'enumeration' types defined in a 'union' type **MUST** be encoded using a CBOR text string data item (major type 3) and **MUST** contain one of the names assigned by 'enum' statements in YANG (see also Section 6.12). The encoding **MUST** be enclosed by the enumeration CBOR tag, as specified in Section 9.3.

Definition example adapted from [RFC7950]:

```
type union {
  type int32;
  type enumeration {
    enum unbounded;
  }
}
```

CBOR diagnostic notation: 44("unbounded")

CBOR encoding: D8 2C 69 756E626F756E646564

6.7. The 'bits' Type

Keeping in mind that bit positions are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in Section 9.7.4.2 of [RFC7950], each element of type bits could be seen as a set of bit positions (or offsets from position 0) that have a value of either 1, which represents the bit being set, or 0, which represents that the bit is not set.

Leafs of type bits **MUST** be encoded either using a CBOR array (major type 4) or byte string (major type 2) or exceptionally as a tagged text string (see below). In case CBOR array representation is used, each element is either (1) a positive integer (major type 0 with value 0 being disallowed) that can be used to calculate the offset of the next byte string or (2) a byte string (major type 2) that carries the information regarding whether certain bits are set or not. The initial offset value

is 0, and each unsigned integer modifies the offset value of the next byte string by the integer value multiplied by 8. For example, if the bit offset is 0 and there is an integer with value 5, the first byte of the byte string that follows will represent bit positions 40 to 47, with both ends included. If the byte string has a second byte, it will carry information about bits 48 to 55, and so on. Within each byte, bits are assigned from least to most significant. After the byte string, the offset is modified by the number of bytes in the byte string multiplied by 8. Bytes with no bits set (zero bytes) at the end of the byte string are never generated. If they occur at the end of the array, the zero bytes are simply omitted; if they occur at the end of a byte string preceding an integer, the zero bytes are removed and the integer is adjusted upwards by the number of zero bytes that were removed. An example follows.

The following example shows the encoding of an 'alarm-state' leaf representation node instance with the 'critical' (position 2), 'warning' (position 8), and 'indeterminate' (position 128) flags set.

```
typedef alarm-state {
  type bits {
    bit unknown;
    bit under-repair;
    bit critical;
    bit major;
    bit minor;
    bit warning {
      position 8;
    }
    bit indeterminate {
      position 128;
    }
  }
}

leaf alarm-state {
  type alarm-state;
}
```

CBOR diagnostic notation: [h'0401', 14, h'01']

CBOR encoding: 83 42 0401 0E 41 01

In a number of cases, the array would only need to have one element -- a byte string with a few bytes inside. For this case, it is **REQUIRED** to omit the array element and have only the byte array that would have been inside. To illustrate this, let us consider the same example YANG definition but this time encoding only 'under-repair' and 'critical' flags. The result would be

CBOR diagnostic notation: h'06'

CBOR encoding: 41 06

Elements in the array **MUST** be either byte strings that do not end in a zero byte or positive unsigned integers, where byte strings and integers **MUST** alternate, i.e., adjacent byte strings or adjacent integers are an error. An array with a single byte string **MUST** instead be encoded as just

that byte string. An array with a single positive integer is an error. Note that a recipient can handle trailing zero bytes in the byte strings using the normal rules without any issue, so an implementation **MAY** silently accept them.

Values of 'bits' types defined in a 'union' type **MUST** be encoded using a CBOR text string data item (major type 3) and **MUST** contain a space-separated sequence of names of 'bits' that are set (see also Section 6.12). The encoding **MUST** be enclosed by the bits CBOR tag, as specified in Section 9.3.

The following example shows the encoding of an 'alarm-state' leaf representation node instance defined using a union type with the 'under-repair' and 'critical' flags set.

Definition example:

```
leaf alarm-state-2 {
  type union {
    type alarm-state;
    type bits {
      bit extra-flag;
    }
  }
}
```

CBOR diagnostic notation: 43("under-repair critical")

CBOR encoding: D8 2B 75 756E6465722D726570616972206372697469746963616C

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 representation node 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 representation node referenced by the 'path' YANG statement.

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

Definition example adapted from [RFC8343]:

```
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: as a YANG Schema Item iDentifier, as defined in Section 3.2, or as a name, as defined in Section 6.8 of [RFC7951]. See Section 6.12 for an exceptional case when this representation needs to be tagged.

6.10.1. SIDs as 'identityref'

When representation nodes of type identityref are implemented using SIDs, they **MUST** be encoded using a CBOR unsigned integer data item (major type 0). (Note that, as they are not used in the position of CBOR map keys, no delta mechanism is employed for SIDs used for identityref.)

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

Definition example adapted 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 Section 3.3. When names are used, identityref **MUST** be encoded using a CBOR text string data item (major type 3). If the identity is defined in a different module than the leaf node containing the identityref data node, the namespace-qualified form **MUST** be used. Otherwise, both the simple and namespace-qualified forms are permitted. Names and namespaces are defined in Section 3.3.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its namespace-qualified 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 an 'is-router' leaf representation node instance when present.

Definition example adapted from [RFC8344]:

```
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 enclosed by a CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 9.3 for the assigned value of these CBOR tags.

As mentioned in Sections 6.6 and in 6.7, 'enumeration' and 'bits' are encoded as a CBOR text string data item (major type 3) when defined within a 'union' type. (This adds considerable complexity but is necessary because of an idiosyncrasy of the YANG data model for unions; the work-around allows compatibility to be maintained with the encoding of overlapping unions in XML and JSON. See also Section 9.12 of [RFC7950].)

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

Definition example adapted from [RFC6991]:

```
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 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, as defined in Section 3.2, and one based on names, as defined in Section 3.3. See Section 6.12 for an exceptional case when this representation needs to be tagged.

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 (representation node). (Note that no delta mechanism is employed for SIDs used for identityref, see Section 6.10.1.)

In the case of a representation node that is an entry of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Instance-identifiers of 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.

Instance-identifiers of representation node entries 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 the top-level list, and followed by each subordinate list(s).

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

Definition example adapted from [RFC7950]:

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

First example:

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

Definition example adapted from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

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

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

This example aims to show how a representation node entry of a YANG list is identified. It uses a somewhat arbitrarily modified YANG module version from [RFC7317] by adding `country` to the leafs and keys of `authorized-key`.

The following example shows the encoding of the 'reporting-entity' value referencing list instance "/system/authentication/user/authorized-key/key-data" (which is assumed to have SID 1734) for username "bob" and authorized-key with name "admin" and country "france".

```
list user {
  key name;

  leaf name {
    type string;
  }

  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key "name country";

    leaf country {
      type string;
    }

    leaf name {
      type string;
    }

    leaf algorithm {
      type string;
    }

    leaf key-data {
      type binary;
    }
  }
}
```

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

CBOR encoding:

```
84                  # array(4)
   19 06C6          # unsigned(1734)
   63               # text(3)
      626F62        # "bob"
   65               # text(5)
      61646D696E    # "admin"
   66               # text(6)
      6672616E6365 # "france"
```

Third example:

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

An 'instance-identifier' value is encoded as a text string that is analogous to the lexical representation in XML encoding; see Section 9.13.2 of [RFC7950]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 3.3, namely:

- The leftmost (top-level) data node name is always in the namespace-qualified form.
- Any subsequent data node name is in the namespace-qualified form if the node is defined in a module other than its parent node; otherwise, the simple form is used. This rule also holds for node names appearing in predicates.

For example,

/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip

is a valid instance-identifier value because the data nodes "interfaces", "interface", and "name" are defined in the module "ietf-interfaces", whereas "ipv4" and "ip" are defined in "ietf-ip".

The resulting XML Path Language (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 1B 2F696574662D73797374656D3A73797374656D2F636F6E74616374
```

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation (the line break is inserted for exposition only):

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

CBOR encoding:

```
78 6B
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A
   65642D6B65795B6E616D653D2761646D696E275D5B636F756E7472793D27
   6672616E6365275D2F6B65792D64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

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

CBOR encoding:

```
78 34                                   # text(52)
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D276A61636B275D
```

7. Content-Types

This specification defines the media type `application/yang-data+cbor`, which can be used without parameters or with the `id` parameter set to either `name` or `sid`.

This media type represents a YANG-CBOR document containing a representation tree. If the media type parameter `id` is present, depending on its value, each representation node is identified by its associated namespace-qualified name, as defined in Section 3.3 (`id=name`), or by its associated YANG SID (represented, e.g., in CBOR map keys as a SID delta or via tag number 47), as defined in Section 3.2 (`id=sid`), respectively. If no `id` parameter is given, both forms may be present.

The format of an `application/yang-data+cbor` representation is that of a CBOR map, mapping names, and/or SIDs (as defined above) into instance values (using the rules defined in Section 4).

It is not foreseen at this point that the valid set of values for the `id` parameter will extend beyond `name`, `sid`, or being unset; if that does happen, any new value is foreseen to be of the form `[a-z][a-z0-9]*(-[a-z0-9]+)*`.

In summary, this document defines three content-types, which are intended for use by different classes of applications:

- `application/yang-data+cbor; id=sid` -- for use by applications that need to be frugal with encoding space and text string processing (e.g., applications running on constrained nodes [RFC7228] or applications with particular performance requirements);
- `application/yang-data+cbor; id=name` -- for use by applications that do not want to engage in SID management and that have ample resources to manage text-string-based item identifiers (e.g., applications that directly want to substitute `application/yang.data+json` with a more efficient representation without any other changes); and
- `application/yang-data+cbor` -- for use by more complex applications that can benefit from the increased efficiency of SID identifiers but also need to integrate databases of YANG modules before SID mappings are defined for them.

All three content-types are based on the same representation mechanisms, parts of which are simply not used in the first and second cases.

How the use of one of these content-types is selected in a transfer protocol is outside the scope of this specification. The last paragraph of Section 5.2 of [RFC8040] discusses how to indicate and request the usage of specific content-types in RESTCONF. Similar mechanisms are available in the Constrained Application Protocol (CoAP) [RFC7252] using the Content-Format and Accept Options; [CORE-COMI] demonstrates specifics on how Content-Format may be used to indicate the `id=sid` case.

8. Security Considerations

The security considerations of [RFC8949] 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 to 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.

For instance, when the `id` parameter to the media type is used, it is important to properly reject identifiers of the other type to avoid scenarios where different implementations interpret a given content in different ways.

When SIDs are in use, the interpretation of encoded data not only relies on having the right YANG modules but also on having the right SID mapping information. Management and evolution of that mapping information therefore requires the same care as the management and evolution of the YANG modules themselves. The procedures in [CORE-SID] are being defined with this in mind.

9. IANA Considerations

9.1. Media Types Registry

IANA has added the following media type to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
yang-data+cbor	application/yang-data+cbor	RFC 9254

Table 2: Media Types Registry

Type name: application

Subtype name: yang-data+cbor

Required parameters: N/A

Optional parameters: id (see Section 7 of RFC 9254)

Encoding considerations: binary (CBOR)

Security considerations: see Section 8 of RFC 9254

Interoperability considerations: N/A

Published specification: RFC 9254

Applications that use this media type: applications that need a concise and efficient representation of YANG-modeled data

Fragment identifier considerations: The syntax and semantics of fragment identifiers specified for "application/yang-data+cbor" is as specified for "application/cbor". (At publication of this document, there is no fragment identification syntax defined for "application/cbor".)

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information: CORE WG mailing list (core@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

Intended usage: COMMON

Restrictions on usage: N/A

Author: CoRE WG

Change controller: IETF

9.2. CoAP Content-Formats Registry

IANA has added the following Content-Formats to the "CoAP Content-Formats" subregistry, within the "Constrained RESTful Environments (CoRE) Parameters" registry [IANA.core-parameters]. The registration procedure is "Expert Review" for the 0-255 range and "IETF Review" for the 256-9999 range.

Media Type	Encoding	ID	Reference
application/yang-data+cbor	-	340	RFC 9254
application/yang-data+cbor; id=name	-	341	RFC 9254
application/yang-data+cbor; id=sid	-	140	RFC 9254

Table 3: CoAP Content-Format Registry

9.3. CBOR Tags Registry

IANA has allocated the following CBOR tag numbers in the "CBOR Tags" registry [IANA.cbor-tags] defined in Section 9.2 of [RFC8949].

Tag	Data Item	Semantics	Reference
43	text string	YANG bits datatype; see Section 6.7.	RFC 9254
44	text string	YANG enumeration datatype; see Section 6.6.	RFC 9254
45	unsigned integer or text string	YANG identityref datatype; see Section 6.10.	RFC 9254
46	unsigned integer or text string or array	YANG instance-identifier datatype; see Section 6.13.	RFC 9254

Tag	Data Item	Semantics	Reference
47	unsigned integer	YANG Schema Item iDentifier (SID); see Section 3.2.	RFC 9254

Table 4: CBOR Tags Registry

10. References

10.1. Normative References

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters/>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types/>.

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[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>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8791] Bierman, A., Björklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/info/rfc8791>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

10.2. Informative References

[CORE-COMI] Veillette, M., Ed., van der Stok, P., Ed., Pelov, A., Bierman, A., and I. Petrov, Ed., "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-11, 17 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-11>.

[CORE-SID] Veillette, M., Ed., Pelov, A., Ed., Petrov, I., Ed., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-18, 18 November 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-18>.

[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>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[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>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

Acknowledgments

This document has been largely inspired by the extensive work done by Andy Bierman and Peter van der Stok on [CORE-COMI]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors of these two documents.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Jürgen Schönwälder and from the Document Shepherd Marco Tiloca. Extensive comments helped us further improve the document in the IESG review process; the authors would like to call out specifically the feedback and guidance by the responsible AD Francesca Palombini and the significant improvements suggested by IESG members Benjamin Kaduk and Rob Wilton.

Authors' Addresses

Michel Veillette (EDITOR)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby Quebec J2J 2V2
Canada
Email: michel.veillette@trilliantinc.com

Ivaylo Petrov (EDITOR)
Google Switzerland GmbH
Brandschenkestrasse 110
CH-8002 Zurich
Switzerland
Email: ivaylopetrov@google.com

Alexander Pelov
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: a@ackl.io

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Michael Richardson
Sandelman Software Works
Canada
Email: mcr+ietf@sandelman.ca