Workgroup: CoRE
Internet-Draft: draft-ietf-core-comi-12
Published: 13 March 2023
Intended Status: Standards Track
Expires: 14 September 2023
Authors: M. V. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. P. Pelov
Acklio
A. Bierman
YumaWorks
C. Bormann, Ed.
Universität Bremen TZI

CoAP Management Interface (CORECONF)

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CORECONF). The Constrained Application Protocol (CoAP) is used to access datastore and data node resources specified in YANG, or SMIv2 converted to YANG. CORECONF uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CORECONF extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-core-comi/.

Discussion of this document takes place on the core Working Group mailing list (mailto:core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/. Subscribe at https://www.ietf.org/mailman/listinfo/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/comi.

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 14 September 2023.

Copyright Notice

Table of Contents

1. Introduction
  1.1. Terminology
2. CORECONF Architecture
  2.1. Major differences between RESTCONF and CORECONF
    2.1.1. Differences due to CoAP and its efficient usage
    2.1.2. Differences due to the use of CBOR
  2.2. Compression of YANG identifiers
  2.3. Instance-identifier
  2.4. Media-Types
  2.5. Unified datastore
3. Example syntax
4. CoAP Interface
  4.1. Using a query parameter for instance identifier keys
  4.2. Data Retrieval
    4.2.1. Using the 'c' query parameter
    4.2.2. Using the 'd' query parameter
    4.2.3. GET
    4.2.4. FETCH
  4.3. Data Editing
    4.3.1. Data Ordering
    4.3.2. POST
    4.3.3. PUT
    4.3.4. iPATCH
    4.3.5. DELETE

4.4. Full datastore access
4.4.1. Full datastore examples
4.5. Event stream
4.5.1. Notify Examples
4.5.2. The 'f' query parameter
4.6. RPC statements
4.6.1. RPC Example
5. Use of Block-wise Transfers
6. Application Discovery
6.1. YANG library
6.2. Resource Discovery
6.2.1. Datastore Resource Discovery
6.2.2. Data node Resource Discovery
6.2.3. Event stream Resource Discovery
7. Error Handling
8. Security Considerations
9. IANA Considerations
9.1. Resource Type (rt=) Link Target Attribute Values Registry
9.2. CoAP Content-Formats Registry
9.3. Media Types Registry
9.4. YANG Namespace Registration
10. References
10.1. Normative References
10.2. Informative References
Appendix A. ietf-coreconf YANG module
Appendix B. ietf-coreconf .sid file
Acknowledgments
Contributors
Authors' Addresses

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city, and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface (CORECONF) which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to [RFC8040] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data models specified in a standardized language, such as YANG, promotes interoperability between devices and applications from different manufacturers.

CORECONF and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs multiple round trips.

To promote small messages, CORECONF uses a YANG to CBOR mapping [RFC9254] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

The following terms are defined in the YANG data modeling language [RFC7950]: action, anydata, anyxml, client, container, data model, data node, identity, instance identifier, leaf, leaf-list, list, module, RPC, schema node, server, submodule.

The following terms are defined in [RFC6241]: configuration data, datastore, state data.

The following term is defined in [I-D.ietf-core-sid]: YANG schema item identifier (YANG SID, often shortened to simply SID).

The following terms are defined in the CoAP protocol [RFC7252]: Confirmable Message, Content-Format, Endpoint.

The following terms are defined in this document:

data node resource: a CoAP resource that models a YANG data node.

datastore resource: a CoAP resource that models a YANG datastore.

event stream resource: a CoAP resource used by clients to observe YANG notifications.

notification instance: An instance of a schema node of type notification, specified in a YANG module implemented by the server. The instance is generated in the server at the occurrence of the corresponding event and reported by an event stream resource.

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list", specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module and data nodes defined

within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance-identifier: List instance identifier or single instance identifier.

instance-value: The value assigned to a data node instance. Instance-values are serialized into the payload according to the rules defined in Section 4 of [RFC9254].

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.

2. CORECONF Architecture

This section describes the CORECONF architecture to use CoAP for reading and modifying the content of datastore(s) used for the management of the instrumented node.

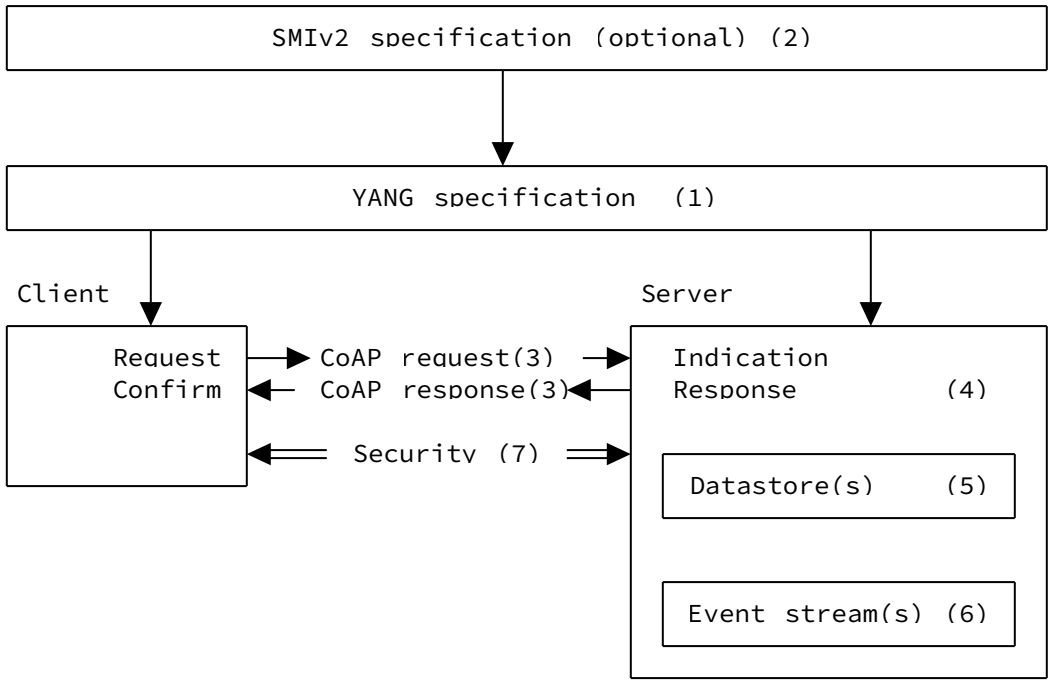

Figure 1: Abstract CORECONF architecture

Figure 1 is a high-level representation of the main elements of the CORECONF management architecture. The different numbered components of Figure 1 are discussed according to the component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: Optional part that consists of a named module which, specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request/response messages: The CORECONF client sends request messages to and receives response messages from the CORECONF server.

(4) Request, Indication, Response, Confirm: Processes performed by the CORECONF clients and servers.

(5) Datastore: A resource used to access configuration data, state data, RPCs, and actions. A CORECONF server may support a single unified datastore or multiple datastores as those defined by Network Management Datastore Architecture (NMDA) [RFC8342].

(6) Event stream: A resource used to get real-time notifications. A CORECONF server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.

(7) Security: The server **MUST** prevent unauthorized users from reading or writing any CORECONF resources. CORECONF relies on security protocols such as DTLS [RFC6347][RFC9147] or OSCORE [RFC8613] to secure CoAP communications.

2.1. Major differences between RESTCONF and CORECONF

CORECONF is a RESTful protocol for small devices where saving bytes to transport a message is very important. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CORECONF is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF.

2.1.1. Differences due to CoAP and its efficient usage

*CORECONF uses CoAP/UDP as transport protocol and CBOR as payload format [RFC9254]. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats.

*CORECONF uses the methods FETCH and iPATCH to access multiple data nodes. RESTCONF uses instead the HTTP method PATCH and the HTTP method GET with the "fields" Query parameter.

*RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not supported by CoAP.

*CORECONF does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.

*CORECONF does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.

2.1.2. Differences due to the use of CBOR

*CORECONF encodes YANG identifier strings as numbers, where RESTCONF does not.

*CORECONF also differs in the handling of default values, only 'report-all' and 'trim' options are supported.

2.2. Compression of YANG identifiers

In the YANG specification, items are identified with a name string. In order to significantly reduce the size of identifiers used in CORECONF, numeric identifiers called YANG Schema Item iDentifier (YANG SID or simply SID) are used instead.

When used in a URI, SIDs are encoded using base64 encoding of the SID bytes. The base64 encoding is using the URL and Filename safe alphabet as defined by Section 5 of [RFC4648], without padding. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (which are essentially leading zeros) at the start of the resulting string are removed. See Figure 2 for complete illustration. (Note that the last paragraph of Section 3.2 of [RFC9254] ensures that SID values being interchanged never can be zero, so at least one base64 "digit" will always remain.)

```
SID in base64 = URLsafeChar[SID >> 60 & 0x3F] |
                URLsafeChar[SID >> 54 & 0x3F] |
                URLsafeChar[SID >> 48 & 0x3F] |
                URLsafeChar[SID >> 42 & 0x3F] |
                URLsafeChar[SID >> 36 & 0x3F] |
                URLsafeChar[SID >> 30 & 0x3F] |
                URLsafeChar[SID >> 24 & 0x3F] |
                URLsafeChar[SID >> 18 & 0x3F] |
                URLsafeChar[SID >> 12 & 0x3F] |
                URLsafeChar[SID >> 6 & 0x3F]  |
                URLsafeChar[SID & 0x3F]
```

Figure 2

For example, SID 1721 is encoded as follows.

```
URLsafeChar[1721 >> 60 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 54 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 48 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 42 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 36 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 30 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 24 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 18 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 12 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 6 & 0x3F]  = URLsafeChar[26] = 'a'
URLsafeChar[1721 & 0x3F]       = URLsafeChar[57] = '5'
```

The resulting base64 representation of SID 1721 is the two-character string "a5".

2.3. Instance-identifier

Instance-identifiers are used to uniquely identify data node instances within a datastore. This YANG built-in type is defined in Section 9.13 of [RFC7950]. An instance-identifier is composed of the data node identifier (i.e. a SID) and, for data nodes within list(s), the keys used to index within these list(s).

When part of a payload, instance-identifiers are encoded in CBOR based on the rules defined in Section 6.13.1 of [RFC9254]. When a (single) instance identifier is part of a URI, the SID is appended as another URI Path component to the URI of the targeted datastore; any keys (further elements of the array specified in Section 6.13.1 of [RFC9254]) are represented using a query parameter as defined in Section 4.1. It would be even simpler to just put the entire 6.13.1 encoding of an instance identifier, base64url-encoded, into a path component of the URI. This would allow FETCH and GET implementations to share almost all of the code. It also would get rid of the Section 2.2 innovation. It would make less obvious which SID is being addressed by a GET during debugging, just as with a FETCH.

2.4. Media-Types

CORECONF uses Media-Types based on the YANG to CBOR mapping specified in [RFC9254].

The following Media-Type is used as defined in [I-D.ietf-core-sid].

*application/yang-data+cbor; id=sid

The following new Media-Types based on CBOR sequences [RFC8742] are defined in this document:

application/yang-identifiers+cbor: This Media-Type represents a CBOR YANG document containing a list of instance-identifiers used to target specific data node instances within a datastore.

FORMAT: CBOR sequence of instance-identifiers

The message payload of Media-Type 'application/yang-identifiers+cbor' is encoded using a CBOR sequence. Each item of this CBOR sequence contains an instance-identifier encoded as defined in Section 6.13.1 of [RFC9254].

application/yang-instances+cbor: This Media-Type represents a CBOR YANG document containing a list of data node instances. Each data node instance is identified by its associated instance-identifier.

FORMAT: CBOR sequence of CBOR maps of instance-identifier, instance-value

The message payload of Media-Type 'application/yang-instances+cbor' is encoded using a CBOR sequence. Each item within this CBOR sequence contains a CBOR map carrying an instance-identifier and associated instance-value. Instance-identifiers are encoded using the rules defined in Section 6.13.1 of [RFC9254], instance-values are encoded using the rules defined in Section 4 of [RFC9254].

When present in an iPATCH request payload, this Media-Type carry a list of data node instances to be replaced, created, or deleted. For each data node instance D, for which the instance-identifier is the same as a data node instance I, in the targeted datastore resource: the value of D replaces the value of I. When the value of D is null, the data node instance I is removed. When the targeted datastore resource does not contain a data node instance with the same instance-identifier as D, a new instance is created with the same instance-identifier and value as D.

The different Media-Type usages are summarized in the table below:

Method	Resource	Media-Type
GET response	data node	application/yang-data+cbor; id=sid
PUT request	data node	application/yang-data+cbor; id=sid
POST request	data node	application/yang-data+cbor; id=sid
DELETE	data node	n/a
GET response	datastore	application/yang-data+cbor; id=sid
PUT request	datastore	application/yang-data+cbor; id=sid
POST request	datastore	application/yang-data+cbor; id=sid
FETCH request	datastore	application/yang-identifiers+cbor

Method	Resource	Media-Type
FETCH response	datastore	application/yang-instances+cbor
iPATCH request	datastore	application/yang-instances+cbor
GET response	event stream	application/yang-instances+cbor
POST request	rpc, action	application/yang-data+cbor; id=sid
POST response	rpc, action	application/yang-data+cbor; id=sid

Table 1

2.5. Unified datastore

CORECONF supports a simple datastore model consisting of a single unified datastore. This datastore provides access to both configuration and operational data. Configuration updates performed on this datastore are reflected immediately or with a minimal delay as operational data.

Alternatively, CORECONF servers **MAY** implement a more complex datastore model such as the Network Management Datastore Architecture (NMDA) as defined by [RFC8342]. Each datastore supported is implemented as a datastore resource.

Characteristics of the unified datastore are summarized in the table below:

Name	Value
Name	unified
YANG modules	all modules
YANG nodes	all data nodes ("config true" and "config false")
Access	read-write
How applied	changes applied in place immediately or with a minimal delay
Protocols	CORECONF
Defined in	"ietf-coreconf"

Table 2

3. Example syntax

CBOR is used to encode CORECONF request and response payloads. The CBOR syntax of the YANG payloads is specified in [RFC9254], based on [RFC8949] and [RFC8742]. The payload examples are notated in Diagnostic notation (defined in Section 8 of [RFC8949]) that can be automatically converted to CBOR.

SIDs in URIs are represented as a base64 number, SIDs in the payload are shown as decimal numbers.

4. CoAP Interface

This document specifies a Management Interface. CoAP endpoints that implement the CORECONF management protocol, support at least one discoverable management resource of resource type (rt): core.c.ds. The path of the discoverable management resource is left to implementers to select (see Section 6).

The mapping of YANG data node instances to CORECONF resources is as follows. Every data node of the YANG modules loaded in the CORECONF server represents a sub-resource of the datastore resource (e.g. /c/YANGSID). When multiple instances of a list exist, instance selection is possible as described in Section 4.1, Section 4.2.3.1, and Section 4.2.4.

CORECONF also supports event stream resources used to observe notification instances. Event stream resources can be discovered using resource type (rt): core.c.ev.

The description of the CORECONF management interface is shown in the table below:

CoAP resource	Example path	rt
Datastore resource	/c	core.c.ds
Data node resource	/c/YANGSID	core.c.dn
Default event steam resource	/s	core.c.ev

Table 3

The path values in the table are example ones. On discovery, the server makes the actual path values known for these resources.

The methods used by CORECONF are:

Operation	Description
GET	Retrieve the datastore resource or a data node resource
FETCH	Retrieve specific data nodes within a datastore resource
POST	Create a datastore resource or a data node resource, invoke an RPC or action
PUT	Create or replace a datastore resource or a data node resource
iPATCH	Idempotently create, replace, and delete data node resource(s) within a datastore resource
DELETE	Delete a datastore resource or a data node resource

Table 4

There is at most one instance of the query parameter for instance identifier keys Section 4.1 for YANG list element selection for the

GET, PUT, POST, and DELETE methods. Having multiple instances of that query parameter shall be treated as an error.

Query parameter	Description
k	Select an instance within YANG list(s)

Table 5

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

4.1. Using a query parameter for instance identifier keys

A query parameter in a URI that does not contain an equals sign specifies a specific instance of a data node. The SID in the URI path is followed by a question mark and the query parameter, which is built from the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

* For a given list, if a parent data node is a list, generate the keys for the parent list first.

* For a given list, generate key values in the order specified in the YANG module.

Key values are first encoded as a CBOR sequence [RFC8742], taking the items defined as array elements for the "following" elements as per the rules of encoding the instance identifier Section 6.13.1 of [RFC9254]. It would be even simpler to just put the entire 6.13.1 encoding of an instance identifier, base64url-encoded, into a path component of the URI. This would allow FETCH and GET implementations to share almost all of the code. It also would get rid of the Section 2.2 innovation. It would make less obvious which SID is being addressed by a GET during debugging, just as with a FETCH. The CBOR sequence is then base64-encoded, using the URL and Filename safe alphabet as defined by Section 5 of [RFC4648], without padding (base64url encoding). The encoder **MUST** ensure, and the decoder **MUST** check, that any unused bits in the last character of that encoding are zero, as per the third sentence of the paragraph after Table 1 in Section 4 of [RFC4648].

4.2. Data Retrieval

One or more data nodes can be retrieved by the client. The operation is mapped to the GET method defined in Section 5.8.1 of [RFC7252] and to the FETCH method defined in Section 2 of [RFC8132].

There are two additional query parameters for the GET and FETCH methods.

query parameters	Description
c	Control selection of configuration and non-configuration data nodes (GET and FETCH)
d	Control retrieval of default values.

Table 6

4.2.1. Using the 'c' query parameter

The 'c' (content) option controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

Table 7

This option is only allowed for GET and FETCH methods on datastore and data node resources. A 4.02 (Bad Option) error is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "a" (the quotes are added for readability, but they are not part of the payload).

4.2.2. Using the 'd' query parameter

The 'd' (with-defaults) option controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported. Defined as 'report-all' in Section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in Section 3.2 of [RFC6243].

Table 8

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value by any client yet, the server **MUST** return the default value of the leaf.

If the target of a GET method is a data node that represents a container or list that has child resources with default values, and these have not been given a value yet,

The server **MUST NOT** return the child resource if d=t

The server **MUST** return the child resource if d=a.

If this query parameter is not present, the default value is "t" (the quotes are added for readability, but they are not part of the payload).

4.2.3. GET

A request to read the value of a data node instance is sent with a CoAP GET message. The URI is set to the data node resource requested, the query parameter for instance identifier keys Section 4.1 is added if any of the parents of the requested data node is a list node.

```
FORMAT:
  GET <data node resource> [Uri-Query option]

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value
```

The returned payload contains the CBOR encoding of the requested instance-value.

4.2.3.1. GET Examples

Using, for example, the current-datetime leaf from module ietf-system [RFC7317], a request is sent to retrieve the value of 'system-state/clock/current-datetime'. The SID of 'system-state/clock/current-datetime' is 1723, encoded in base64 according to Section 2.2, yields a7. The response to the request returns the CBOR map with the key set to the SID of the requested data node (i.e. 1723) and the value encoded using a 'text string' as defined in Section 4 of [RFC9254]. The datastore resource path /c is an example location discovered with a request similar to Figure 4.

```
REQ: GET </c/a7>

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1723 : "2014-10-26T12:16:31Z"
}
```

The next example represents the retrieval of a YANG container. In this case, the CORECONF client performs a GET request on the clock container (SID = 1721; base64: a5). The container returned is encoded using a CBOR map as specified by Section 4.2 of [RFC9254].

```
REQ: GET </c/a5>

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1721 : {
    2 : "2014-10-26T12:16:51Z",    / current-datetime (SID 1723) /
    1 : "2014-10-21T03:00:00Z"     / boot-datetime (SID 1722) /
  }
}
```

Figure 3

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded employing a CBOR array as specified by Section 4.4.1 of [RFC9254] containing 2 list entries.

```
REQ: GET </c/X9>

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",                 / name  (SID 1537) /
      1 : "Ethernet adaptor",     / description (SID 1534) /
      5 : 1880,                   / type, (SID 1538) identity /
                                  / ethernetCsmacd (SID 1880) /
      2 : true                    / enabled (SID 1535) /
    },
    {
      4 : "eth1",                 / name (SID 1537) /
      1 : "Ethernet adaptor",     / description (SID 1534) /
      5 : 1880,                   / type, (SID 1538) identity /
                                  / ethernetCsmacd (SID 1880) /
      2 : false                   / enabled (SID 1535) /
    }
  ]
}
```

To retrieve a specific entry within the /interfaces/interface YANG list, the CORECONF client adds the key of the targeted instance in its CoAP request using the query parameter for instance identifier

keys Section 4.1. The return payload containing the instance requested is encoded using a CBOR array as specified by Section 4.4.1 of [RFC9254] containing the requested entry.

```
REQ: GET </c/X9?ZGV0aDA>        [1533, "eth0"]

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",                  / name  (SID 1537) /
      1 : "Ethernet adaptor",      / description (SID 1534) /
      5 : 1880,                    / type, (SID 1538) identity /
                                   / ethernetCsmacd (SID 1880) /
      2 : true                     / enabled (SID 1535) /
    }
  ]
}
```

It is equally possible to select a leaf of a specific entry of a list. The example below requests the description leaf (SID 1534, base64: X-) within the interface list corresponding to the interface name "eth0". The returned value is encoded in CBOR based on the rules specified by Section 6.4 of [RFC9254].

```
REQ: GET </c/X-?ZGV0aDA>        [1534, "eth0"]

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1534 : "Ethernet adaptor"
}
```

4.2.4. FETCH

The FETCH is used to retrieve multiple instance-values. The FETCH request payload contains the list of instance-identifiers of the data node instances requested.

The return response payload contains a list of data node instance-values in the same order as requested. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

For compactness, indexes of the list instance identifiers returned by the FETCH response **SHOULD** be elided, only the SID is provided. This approach may also help reducing implementation complexity since the format of each entry within the CBOR sequence of the FETCH

response is identical to the format of the corresponding GET response.

```
FORMAT:
  FETCH <datastore resource>
        (Content-Format: application/yang-identifiers+cbor)
  CBOR sequence of instance-identifiers

  2.05 Content (Content-Format: application/yang-instances+cbor)
  CBOR sequence of CBOR maps of SID, instance-value
```

4.2.4.1. FETCH examples

This example uses the current-datetime leaf from module ietf-system [RFC7317] and the interface list from module ietf-interfaces [RFC8343]. In this example the value of current-datetime (SID 1723) and the interface list (SID 1533) instance identified with name="eth0" are queried.

```
REQ: FETCH </c>
     (Content-Format: application/yang-identifiers+cbor)
1723,            / current-datetime (SID 1723) /
[1533, "eth0"]   / interface (SID 1533) with name = "eth0" /

RES: 2.05 Content (Content-Format: application/yang-instances+cbor)

{
  1723 : "2014-10-26T12:16:31Z" / current-datetime (SID 1723) /
},
{
  1533 : {
     4 : "eth0",              / name (SID 1537) /
     1 : "Ethernet adaptor",  / description (SID 1534) /
     5 : 1880,                / type (SID 1538), identity /
                              / ethernetCsmacd (SID 1880) /
     2 : true,                / enabled (SID 1535) /
    11 : 3             / oper-status (SID 1544), value is testing /
  }
}
```

4.3. Data Editing

CORECONF allows datastore contents to be created, modified and deleted using CoAP methods.

4.3.1. Data Ordering

A CORECONF server **MUST** preserve the relative order of all user-ordered list and leaf-list entries that are received in a single

edit request. As per [RFC9254], these YANG data node types are encoded as CBOR arrays, so messages will preserve their order.

4.3.2. POST

The CoAP POST operation is used in CORECONF for the creation of data node resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 4.6 for details on "ACTION" and "RPC" resources.

A request to create a data node instance is sent with a CoAP POST message. The URI specifies the data node resource of the instance to be created. In the case of a list instance, keys **MUST** be present in the payload.

```
FORMAT:
  POST <data node resource>
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created
```

If the data node instance already exists, then the POST request **MUST** fail and a "4.09 Conflict" response code **MUST** be returned

4.3.2.1. Post example

The example uses the interface list from module ietf-interfaces [RFC8343]. This example creates a new list instance within the interface list (SID = 1533), while assuming the datastore resource is hosted on the CoAP server with DNS name example.com and with path /ds. The path /ds is an example location that is assumed to have been discovered using request similar to Figure 4.

```
REQ: POST <coap://example.com/ds/X9>
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth5",              / name (SID 1537) /
      1 : "Ethernet adaptor",  / description (SID 1534) /
      5 : 1880,                / type (SID 1538), identity /
                               / ethernetCsmacd (SID 1880) /
      2 : true                 / enabled (SID 1535) /
    }
  ]
}

RES: 2.01 Created
```

4.3.3. PUT

A data node resource instance is created or replaced with the PUT method. A request to set the value of a data node instance is sent with a CoAP PUT message.

```
FORMAT:
  PUT <data node resource> [Uri-Query option]
      (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created
```

4.3.3.1. PUT example

The example uses the interface list from module ietf-interfaces [RFC8343]. This example updates the instance of the list interface (SID = 1533) with key name="eth0". The example location /c is an example location that is discovered using a request similar to Figure 4.

```
REQ: PUT </c/X9?ZGV0aDA>       [1533, "eth0"]
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1533 : [
    {
      4 : "eth0",              / name (SID 1537) /
      1 : "Ethernet adaptor",  / description (SID 1534) /
      5 : 1880,                / type (SID 1538), identity /
                               / ethernetCsmacd (SID 1880) /
      2 : true                 / enabled (SID 1535) /
    }
  ]
}

RES:  2.04 Changed
```

4.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent CoAP iPATCH method [RFC8132].

There are no query parameters for the iPATCH method.

The processing of the iPATCH command is specified by Media-Type 'application/yang-instances+cbor'. In summary, if the CBOR patch payload contains a data node instance that is not present in the target, this instance is added. If the target contains the specified instance, the content of this instance is replaced with the value of

the payload. A null value indicates the removal of an existing data node instance.

```
FORMAT:
  iPATCH <datastore resource>
         (Content-Format: application/yang-instances+cbor)
  CBOR sequence of CBOR maps of instance-identifier, instance-value

  2.04 Changed
```

4.3.4.1. iPATCH example

In this example, a CORECONF client requests the following operations:

- *Set "/system/ntp/enabled" (SID 1755) to true.
- *Remove the server "tac.nrc.ca" from the "/system/ntp/server" (SID 1756) list.
- *Add/set the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1756).

```
REQ: iPATCH </c>
     (Content-Format: application/yang-instances+cbor)
{
  1755 : true                   / enabled (SID 1755) /
},
{
  [1756, "tac.nrc.ca"] : null   / server (SID 1756) /
},
{
  1756 : {                      / server (SID 1756) /
    3 : "tic.nrc.ca",           / name (SID 1759) /
    4 : true,                   / prefer (SID 1760) /
    5 : {                       / udp (SID 1761) /
      1 : "132.246.11.231"      / address (SID 1762) /
    }
  }
}

RES: 2.04 Changed
```

4.3.5. DELETE

A data node resource is deleted with the DELETE method.

```
FORMAT:
  Delete <data node resource> [Uri-Query option]

  2.02 Deleted
```

4.3.5.1. DELETE example

This example uses the interface list from module ietf-interfaces [RFC8343]. This example deletes an instance of the interface list (SID = 1533):

```
REQ:   DELETE </c/X9?ZGV0aDA>        [1533, "eth0"]

RES:   2.02 Deleted
```

4.4. Full datastore access

The methods GET, PUT, POST, and DELETE can be used to request, replace, create, and delete a whole datastore respectively.

```
FORMAT:
  GET <datastore resource>

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

FORMAT:
  PUT <datastore resource>
      (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.04 Changed

FORMAT:
  POST <datastore resource>
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.01 Created

FORMAT:
  DELETE <datastore resource>

  2.02 Deleted
```

The content of the CBOR map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request.

4.4.1. Full datastore examples

The example uses the interface list from module ietf-interfaces [RFC8343] and the clock container from module ietf-system [RFC7317]. We assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the 'interface' list (SID 1533) with one instance and the 'clock' container (SID 1721). After invocation of GET, a CBOR map with data nodes from these two modules is returned:

```
REQ:  GET </c>

RES: 2.05 Content
     (Content-Format: application/yang-data+cbor; id=sid)
{
  1721 : {                      / Clock (SID 1721) /
    2: "2016-10-26T12:16:31Z",  / current-datetime (SID 1723) /
    1: "2014-10-05T09:00:00Z"   / boot-datetime (SID 1722) /
  },
  1533 : [
    {                           / interface (SID 1533) /
       4 : "eth0",              / name (SID 1537) /
       1 : "Ethernet adaptor",  / description (SID 1534) /
       5 : 1880,                / type (SID 1538), identity: /
                                / ethernetCsmacd (SID 1880) /
       2 : true,                / enabled (SID 1535) /
      11 : 3             / oper-status (SID 1544), value is testing /
    }
  ]
}
```

4.5. Event stream

Event notification is an essential function for the management of servers. CORECONF allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The path for the default event stream can be discovered as described in Section 4. The server **MAY** support additional event stream resources to address different notification needs.

Reception of notification instances is enabled with the CoAP Observe [RFC7641] function. Clients subscribe to the notifications by sending a GET request with an "Observe" option to the stream resource.

Each response payload carries one or multiple notifications. The number of notifications reported, and the conditions used to remove notifications from the reported list are left to implementers. When multiple notifications are reported, they **MUST** be ordered starting

from the newest notification at index zero. Note that this could lead to notifications being sent multiple times, which increases the probability for the client to receive them, but it might potentially lead to messages that exceed the MTU of a single CoAP packet. If such cases could arise, implementers should make sure appropriate fragmentation is available - for example the one described in Section 5.

The format of notifications is a CBOR sequence, where each item in the sequence is a single notification as described in Section 4.2.1 of [RFC9254]. (Accordingly, a notification without any content is an empty CBOR sequence, i.e., zero bytes.)

```
FORMAT:
  GET <stream-resource> Observe(0)

  2.05 Content (Content-Format: application/yang-instances+cbor)
  CBOR sequence of CBOR maps of instance-identifier, instance-value
```

The sequence of data node instances may contain identical items which have been generated at different times.

An example implementation is:

> Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After an aggregation period, which may be limited by the maximum number of notifications supported, the content of the instance is sent to all clients observing the modified stream.

4.5.1. Notify Examples

Let suppose the server generates the example-port-fault event as defined below.

```
module example-port {
  ...
  notification example-port-fault {   // SID 60010
    description
      "Event generated if a hardware fault is detected";
    leaf port-name {                  // SID 60011
      type string;
    }
    leaf port-fault {                 // SID 60012
      type string;
    }
  }
}
```

In this example the default event stream resource path /s is an example location discovered with a request similar to Figure 5. By executing a GET with Observe 0 on the default event stream resource the client receives the following response:

```
REQ:  GET </s> Observe(0)

RES:  2.05 Content (Content-Format: application/yang-instances+cbor)
      Observe(12)

{
  60010 : {             / example-port-fault (SID 60010) /
    1 : "0/4/21",       / port-name (SID 60011) /
    2 : "Open pin 2"    / port-fault (SID 60012) /
  }
},
{
  60010 : {             / example-port-fault (SID 60010) /
    1 : "1/4/21",       / port-name (SID 60011) /
    2 : "Open pin 5"    / port-fault (SID 60012) /
  }
}
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

4.5.2. The 'f' query parameter

The 'f' (filter) option is used to indicate which subset of all possible notifications is of interest. If not present, all notifications supported by the event stream are reported.

When not supported by a CORECONF server, this option shall be ignored, all events notifications are reported independently of the presence and content of the 'f' (filter) option.

When present, this option contains a comma-separated list of notification SIDs. For example, the following request returns notifications 60010 and 60020.

```
REQ:  GET </s?f=60010,60020> Observe(0)
```

4.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote Procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance.

The request payload contains the values assigned to the input container when specified. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in Section 4.2.1 of [RFC9254].

The returned success response code is 2.05 Content.

```
FORMAT:
  POST <data node resource> [Uri-Query option]
       (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value

  2.05 Content (Content-Format: application/yang-data+cbor; id=sid)
  CBOR map of SID, instance-value
```

4.6.1. RPC Example

The example is based on the YANG action "reset" as defined in Section 7.15.3 of [RFC7950] and annotated below with SIDs.

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {                        // SID 60000
    key name;
    leaf name {                        // SID 60001
      type string;
    }
    action reset {                     // SID 60002
      input {
        leaf reset-at {                // SID 60003
          type yang:date-and-time;
          mandatory true;
        }
       }
       output {
         leaf reset-finished-at {      // SID 60004
           type yang:date-and-time;
           mandatory true;
         }
       }
     }
   }
 }
```

This example invokes the 'reset' action (SID 60002, base64: Opq), of the server instance with name equal to "myserver".

```
REQ:  POST </c/Opq?aG15c2VydmVy>     [60002, "myserver"]
      (Content-Format: application/yang-data+cbor; id=sid)
{
  60002 : {
    1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
  }
}

RES:  2.05 Content
      (Content-Format: application/yang-data+cbor; id=sid)
{
  60002 : {
    2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
  }
}
```

5. Use of Block-wise Transfers

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of data need to be transported, datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process complete data fields.

Beware of race conditions. In case blocks are filled one at a time, care should be taken that the whole and consistent data representation is sent in multiple blocks sequentially without interruption. On the server, values might change, lists might get re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the resource, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with the actual number of items. It may be advisable to use Indefinite-length CBOR arrays and maps, which are foreseen for data streaming purposes. (Note that the outer structure of yang-identifiers and yang-instances is a CBOR sequence, which already behaves similar to an indefinite-length encoded array.)

6. Application Discovery

Two application discovery mechanisms are supported by CORECONF, the YANG library data model as defined by [I-D.ietf-core-yang-library] and the CORE resource discovery [RFC6690]. Implementers may choose to implement one or the other or both.

6.1. YANG library

The YANG library data model [I-D.ietf-core-yang-library] provides a high-level description of the resources available. The YANG library contains the list of modules, features, and deviations supported by the CORECONF server. From this information, CORECONF clients can infer the list of data nodes supported and the interaction model to be used to access them. This module also contains the list of datastores implemented.

As described in [RFC6690], the location of the YANG library can be found by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.yl". Upon success, the return payload will contain the root resource of the YANG library module.

The following example assumes that the SID of the YANG library is 2351 (kv after encoding as specified in Section 2.2) and that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.yl>

RES: 2.05 Content (Content-Format: application/link-format)
</c/kv>;rt="core.c.yl"
```

6.2. Resource Discovery

As some CoAP interfaces and services might not support the YANG library interface and still be interested to discover resources that are available, implementations **MAY** choose to support discovery of all available resources using "/.well-known/core" as defined by [RFC6690].

6.2.1. Datastore Resource Discovery

The presence and location of (path to) each datastore implemented by the CORECONF server can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.ds".

Upon success, the return payload contains the list of datastore resources.

Each datastore returned is further qualified using the "ds" Link-Format attribute. This attribute is set to the SID assigned to the datastore identity. When a unified datastore is implemented, the ds attribute is set to 1029 as specified in Appendix B. For other examples of datastores, see the Network Management Datastore Architecture (NMDA) [RFC7950].

```
link-extension    = ( "ds" "=" sid ) )
                    ; SID assigned to the datastore identity
sid               = 1*DIGIT
```

The following example assumes that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.ds>

RES: 2.05 Content (Content-Format: application/link-format)
</c>; rt="core.c.ds";ds=1029
```

Figure 4

6.2.2. Data node Resource Discovery

If implemented, the presence and location of (path to) each data node implemented by the CORECONF server are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.dn".

Upon success, the return payload contains the SID assigned to each data node and their location.

The example below shows the discovery of the presence and location of data nodes. Data nodes '/ietf-system:system-state/clock/boot-datetime' (SID 1722) and '/ietf-system:system-state/clock/current-datetime' (SID 1723) are returned. The example assumes that the server uses /c as datastore resource path.

```
REQ: GET </.well-known/core?rt=core.c.dn>

RES: 2.05 Content (Content-Format: application/link-format)
</c/a6>;rt="core.c.dn",
</c/a7>;rt="core.c.dn"
```

Without additional filtering, the list of data nodes may become prohibitively long. If this is the case implementations **SHOULD** support a way to obtain all links using multiple GET requests (for example through some form of pagination).

6.2.3. Event stream Resource Discovery

The presence and location of (path to) each event stream implemented by the CORECONF server are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.es".

Upon success, the return payload contains the list of event stream resources.

The following example assumes that the server uses /s as the default event stream resource.

```
REQ: GET </.well-known/core?rt=core.c.es>

RES: 2.05 Content (Content-Format: application/link-format)
</s>;rt="core.c.es"
```

Figure 5

7. Error Handling

In case a request is received which cannot be processed properly, the CORECONF server **MUST** return an error response. This error response **MUST** contain a CoAP 4.xx or 5.xx response code.

Errors returned by a CORECONF server can be broken into two categories, those associated with the CoAP protocol itself and those generated during the validation of the YANG data model constraints as described in Section 8 of [RFC7950].

The following list of common CoAP errors should be implemented by CORECONF servers. This list is not exhaustive, other errors defined by CoAP and associated RFCs may be applicable.

* Error 4.01 (Unauthorized) is returned by the CORECONF server when the CORECONF client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).

* Error 4.02 (Bad Option) is returned by the CORECONF server when one or more CoAP options are unknown or malformed.

* Error 4.04 (Not Found) is returned by the CORECONF server when the CORECONF client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).

* Error 4.05 (Method Not Allowed) is returned by the CORECONF server when the CORECONF client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).

* Error 4.08 (Request Entity Incomplete) is returned by the CORECONF server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.

* Error 4.13 (Request Entity Too Large) may be returned by the CORECONF server during a block transfer request, see [RFC7959] for more details.

* Error 4.15 (Unsupported Content-Format) is returned by the CORECONF server when the Content-Format used in the request does not match those specified in Section 2.4.

The CORECONF server **MUST** also enforce the different constraints associated with the YANG data models implemented. These constraints are described in Section 8 of [RFC7950]. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using the encoding rules of a YANG data template as defined in Section 5 of [RFC9254].

```
+--rw error!
   +--rw error-tag             identityref
   +--rw error-app-tag?        identityref
   +--rw error-data-node?      instance-identifier
   +--rw error-message?        string
```

The following 'error-tag' and 'error-app-tag' are defined by the ietf-coreconf YANG module, these tags are implemented as YANG identity and can be extended as needed.

*error-tag 'operation-failed' is returned by the CORECONF server when the operation request cannot be processed successfully.

-error-app-tag 'malformed-message' is returned by the CORECONF server when the payload received from the CORECONF client does not contain a well-formed CBOR content as defined in [RFC8949] or does not comply with the CBOR structure defined within this document.

-error-app-tag 'data-not-unique' is returned by the CORECONF server when the validation of the 'unique' constraint of a list or leaf-list fails.

-error-app-tag 'too-many-elements' is returned by the CORECONF server when the validation of the 'max-elements' constraint of a list or leaf-list fails.

-error-app-tag 'too-few-elements' is returned by the CORECONF server when the validation of the 'min-elements' constraint of a list or leaf-list fails.

-error-app-tag 'must-violation' is returned by the CORECONF server when the restrictions imposed by a 'must' statement are violated.

-error-app-tag 'duplicate' is returned by the CORECONF server when a client tries to create a duplicate list or leaf-list entry.

*error-tag 'invalid-value' is returned by the CORECONF server when the CORECONF client tries to update or create a leaf with a value

encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.

- -error-app-tag 'invalid-datatype' is returned by the CORECONF server when CBOR encoding does not follow the rules set by the YANG Build-In type or when the value is incompatible with it (e.g. a value greater than 127 for an int8, undefined enumeration).

- -error-app-tag 'not-in-range' is returned by the CORECONF server when the validation of the 'range' property fails.

- -error-app-tag 'invalid-length' is returned by the CORECONF server when the validation of the 'length' property fails.

- -error-app-tag 'pattern-test-failed' is returned by the CORECONF server when the validation of the 'pattern' property fails.

* error-tag 'missing-element' is returned by the CORECONF server when the operation requested by a CORECONF client fails to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.

- -error-app-tag 'missing-key' is returned by the CORECONF server to further qualify a missing-element error. This error is returned when the CORECONF client tries to create or list instance, without all the 'key' specified or when the CORECONF client tries to delete a leaf listed as a 'key'.

- -error-app-tag 'missing-input-parameter' is returned by the CORECONF server when the input parameters of an RPC or action are incomplete.

* error-tag 'unknown-element' is returned by the CORECONF server when the CORECONF client tries to access a data node of a YANG module not supported, of a data node associated with an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

* error-tag 'bad-element' is returned by the CORECONF server when the CORECONF client tries to create data nodes for more than one case in a choice.

*error-tag 'data-missing' is returned by the CORECONF server when a data node required to accept the request is not present.

-error-app-tag 'instance-required' is returned by the CORECONF server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.

-error-app-tag 'missing-choice' is returned by the CORECONF server when no nodes exist in a mandatory choice.

*error-tag 'error' is returned by the CORECONF server when an unspecified error has occurred.

For example, the CORECONF server might return the following error.

```
RES:  4.00 Bad Request
     (Content-Format: application/yang-data+cbor; id=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 value exceeded" / error-message (SID 1027) /
  }
}
```

8. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CORECONF re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347][RFC9147] and OSCORE [RFC8613] for protected access to resources, as well as suitable authentication and authorization mechanisms, for example those defined in ACE OAuth [RFC9200].

All the security considerations of [RFC7252], [RFC7959], [RFC8132] and [RFC7641] apply to this document as well. The use of NoSec (Section 9 of [RFC7252]), when OSCORE is not used, is **NOT RECOMMENDED**.

In addition, mechanisms for authentication and authorization may need to be selected if not provided with the CoAP security mode.

As [RFC9254] and [RFC4648] are used for payload and SID encoding, the security considerations of those documents also need to be well-understood.

9. IANA Considerations

9.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Value	Description	Reference
core.c.ds	YANG datastore	RFC XXXX
core.c.dn	YANG data node	RFC XXXX
core.c.yl	YANG module library	RFC XXXX
core.c.es	YANG event stream	RFC XXXX

Table 9

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

Media Type	Content Coding	ID	Reference
application/yang-identifiers+cbor		TBD2	RFC XXXX
application/yang-instances+cbor		TBD3	RFC XXXX

Table 10

// RFC Ed.: replace TBD1, TBD2 and TBD3 with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

Name	Template	Reference
yang-identifiers+cbor	application/	RFC XXXX
	yang-identifiers+cbor	
yang-instances+cbor	application/	RFC XXXX
	yang-instances+cbor	

Table 11

Each of these media types share the following information:

*Subtype name: <as listed in table>

*Required parameters: N/A

*Optional parameters: N/A

*Encoding considerations: binary

*Security considerations: See the Security Considerations section of RFC XXXX

*Interoperability considerations: N/A

*Published specification: RFC XXXX

*Applications that use this media type: CORECONF

*Fragment identifier considerations: N/A

*Additional information:

* Deprecated alias names for this type: N/A

* 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: iesg&ietf.org

*Intended usage: COMMON

*Restrictions on usage: N/A

*Author: Michel Veillette

*Change Controller: IESG

*Provisional registration? No

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.4. YANG Namespace Registration

This document registers the following XML namespace URN in the "IETF XML Registry", following the format defined in [RFC3688]:

URI: please assign urn:ietf:params:xml:ns:yang:ietf-coreconf

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

Reference: RFC XXXX

// RFC Ed.: please replace XXXX with RFC number and remove this note

10. References

10.1. Normative References

[I-D.ietf-core-sid] Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-20, 1 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-20>.

[I-D.ietf-core-yang-library] Veillette, M. and I. Petrov, "Constrained YANG Module Library", Work in Progress, Internet-Draft, draft-ietf-core-yang-library-03, 11 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-yang-library-03>.

[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/rfc/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/rfc/rfc5277>.

[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/rfc/rfc6241>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <https://www.rfc-editor.org/rfc/rfc6243>.

[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/rfc/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

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

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/rfc/rfc7959>.

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/rfc/rfc8132>.

[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/rfc/rfc8174>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/rfc/rfc8742>.

[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/rfc/rfc8949>.

[RFC9254] Veillette, M., Ed., Petrov, I., Ed., Pelov, A., Bormann, C., and M. Richardson, "Encoding of Data Modeled with YANG in the Concise Binary Object Representation (CBOR)", RFC 9254, DOI 10.17487/RFC9254, July 2022, <https://www.rfc-editor.org/rfc/rfc9254>.

10.2. Informative References

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/rfc/rfc6347>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

[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/rfc/rfc7317>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/rfc/rfc8342>.

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

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

Appendix A. ietf-coreconf YANG module

```
<CODE BEGINS> file "ietf-coreconf@2023-03-13.yang"

module ietf-coreconf {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-coreconf";
  prefix coreconf;

  import ietf-datastores {
    prefix ds;
  }

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is required to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
     <mailto:michel.veillette@trilliantinc.com>

     Alexander Pelov
     <mailto:alexander@ackl.io>

     Peter van der Stok
     <mailto:consultancy@vanderstok.org>

     Andy Bierman
     <mailto:andy@yumaworks.com>";

  description
    "This module contains the different definitions required
     by the CORECONF protocol.

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

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (https://trustee.ietf.org/license-info).
```

```
   This version of this YANG module is part of RFC XXXX;
   see the RFC itself for full legal notices.";

revision 2023-03-13 {
   description
    "Initial revision.";
  reference
    "[I-D.ietf-core-comi] CoAP Management Interface";
}

identity unified {
  base ds:datastore;
  description
    "Identifier of the unified configuration and operational
     state datastore.";
}

identity error-tag {
  description
    "Base identity for error-tag.";
}

identity operation-failed {
  base error-tag;
  description
    "Returned by the CORECONF server when the operation request
     can't be processed successfully.";
}

identity invalid-value {
  base error-tag;
  description
    "Returned by the CORECONF server when the CORECONF client tries
     to update or create a leaf with a value encoded using an
     invalid CBOR datatype or if the 'range', 'length',
     'pattern' or 'require-instance' constrain is not
     fulfilled.";
}

identity missing-element {
  base error-tag;
  description
    "Returned by the CORECONF server when the operation requested
     by a CORECONF client fails to comply with the 'mandatory'
     constraint defined. The 'mandatory' constraint is
     enforced for leafs and choices, unless the node or any of
     its ancestors have a 'when' condition or 'if-feature'
     expression that evaluates to 'false'.";
}
```

```
identity unknown-element {
  base error-tag;
  description
    "Returned by the CORECONF server when the CORECONF client tries
     to access a data node of a YANG module not supported, of a
     data node associated with an 'if-feature' expression
     evaluated to 'false' or to a 'when' condition evaluated
     to 'false'.";
}

identity bad-element {
  base error-tag;
  description
    "Returned by the CORECONF server when the CORECONF client tries
     to create data nodes for more than one case in a choice.";
}

identity data-missing {
  base error-tag;
  description
    "Returned by the CORECONF server when a data node required to
     accept the request is not present.";
}

identity error {
  base error-tag;
  description
    "Returned by the CORECONF server when an unspecified error has
    occurred.";
}

identity error-app-tag {
  description
    "Base identity for error-app-tag.";
}

identity malformed-message {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the payload received
     from the CORECONF client don't contain a well-formed CBOR
     content as defined in [RFC8949] or don't
     comply with the CBOR structure defined within this
     document.";
}

identity data-not-unique {
  base error-app-tag;
```

```
  description
    "Returned by the CORECONF server when the validation of the
     'unique' constraint of a list or leaf-list fails.";
}

identity too-many-elements {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the validation of the
     'max-elements' constraint of a list or leaf-list fails.";
}

identity too-few-elements {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the validation of the
     'min-elements' constraint of a list or leaf-list fails.";
}

identity must-violation {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the restrictions
     imposed by a 'must' statement are violated.";
}

identity duplicate {
  base error-app-tag;
  description
    "Returned by the CORECONF server when a client tries to create
     a duplicate list or leaf-list entry.";
}

identity invalid-datatype {
  base error-app-tag;
  description
    "Returned by the CORECONF server when CBOR encoding is
     incorect or when the value encoded is incompatible with
     the YANG Built-In type. (e.g. value greater than 127
     for an int8, undefined enumeration).";
}

identity not-in-range {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the validation of the
     'range' property fails.";
}
```

```
identity invalid-length {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the validation of the
     'length' property fails.";
}

identity pattern-test-failed {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the validation of the
     'pattern' property fails.";
}

identity missing-key {
  base error-app-tag;
  description
    "Returned by the CORECONF server to further qualify a
     missing-element error. This error is returned when the
     CORECONF client tries to create or list instance, without all
     the 'key' specified or when the CORECONF client tries to
     delete a leaf listed as a 'key'.";
}

identity missing-input-parameter {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the input parameters
     of a RPC or action are incomplete.";
}

identity instance-required {
  base error-app-tag;
  description
    "Returned by the CORECONF server when a leaf of type
     'instance-identifier' or 'leafref' marked with
     require-instance set to 'true' refers to an instance
     that does not exist.";
}

identity missing-choice {
  base error-app-tag;
  description
    "Returned by the CORECONF server when no nodes exist in a
     mandatory choice.";
}

rc:yang-data coreconf-error {
  container error {
```

```
      description
        "Optional payload of a 4.00 Bad Request CoAP error.";

      leaf error-tag {
        type identityref {
          base error-tag;
        }
        mandatory true;
        description
          "The enumerated error-tag.";
      }

      leaf error-app-tag {
        type identityref {
          base error-app-tag;
        }
        description
          "The application-specific error-tag.";
      }

      leaf error-data-node {
        type instance-identifier;
        description
          "When the error reported is caused by a specific data node,
           this leaf identifies the data node in error.";
      }

      leaf error-message {
        type string;
        description
          "A message describing the error.";
      }
    }
  }
}

<CODE ENDS>
```

Appendix B. ietf-coreconf .sid file

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-coreconf",
  "module-revision": "2023-03-13",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-coreconf",
      "sid": 1000
    },
    {
      "namespace": "identity",
      "identifier": "bad-element",
      "sid": 1001
    },
    {
      "namespace": "identity",
      "identifier": "data-missing",
      "sid": 1002
    },
    {
      "namespace": "identity",
      "identifier": "data-not-unique",
      "sid": 1003
    },
    {
      "namespace": "identity",
      "identifier": "duplicate",
      "sid": 1004
    },
    {
      "namespace": "identity",
      "identifier": "error",
      "sid": 1005
    },
    {
      "namespace": "identity",
      "identifier": "error-app-tag",
      "sid": 1006
    },
    {
      "namespace": "identity",
      "identifier": "error-tag",
      "sid": 1007
```

```
},
{
  "namespace": "identity",
  "identifier": "instance-required",
  "sid": 1008
},
{
  "namespace": "identity",
  "identifier": "invalid-datatype",
  "sid": 1009
},
{
  "namespace": "identity",
  "identifier": "invalid-length",
  "sid": 1010
},
{
  "namespace": "identity",
  "identifier": "invalid-value",
  "sid": 1011
},
{
  "namespace": "identity",
  "identifier": "malformed-message",
  "sid": 1012
},
{
  "namespace": "identity",
  "identifier": "missing-choice",
  "sid": 1013
},
{
  "namespace": "identity",
  "identifier": "missing-element",
  "sid": 1014
},
{
  "namespace": "identity",
  "identifier": "missing-input-parameter",
  "sid": 1015
},
{
  "namespace": "identity",
  "identifier": "missing-key",
  "sid": 1016
},
{
  "namespace": "identity",
  "identifier": "must-violation",
```

```
  "sid": 1017
},
{
  "namespace": "identity",
  "identifier": "not-in-range",
  "sid": 1018
},
{
  "namespace": "identity",
  "identifier": "operation-failed",
  "sid": 1019
},
{
  "namespace": "identity",
  "identifier": "pattern-test-failed",
  "sid": 1020
},
{
  "namespace": "identity",
  "identifier": "too-few-elements",
  "sid": 1021
},
{
  "namespace": "identity",
  "identifier": "too-many-elements",
  "sid": 1022
},
{
  "namespace": "identity",
  "identifier": "unified",
  "sid": 1029
},
{
  "namespace": "identity",
  "identifier": "unknown-element",
  "sid": 1023
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error",
  "sid": 1024
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error/error-app-tag",
  "sid": 1025
},
{
  "namespace": "data",
```

```
      "identifier": "/ietf-coreconf:error/error-data-node",
      "sid": 1026
    },
    {
      "namespace": "data",
      "identifier": "/ietf-coreconf:error/error-message",
      "sid": 1027
    },
    {
      "namespace": "data",
      "identifier": "/ietf-coreconf:error/error-tag",
      "sid": 1028
    }
  ]
}
```

Acknowledgments

We are very grateful to Bert Greevenbosch who was one of the original authors of the CORECONF specification.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Koen Zandberg's implementation input motivated massively simplifying (and fixing) the URI construction for GET/PUT/POST requests.

The draft has further benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Klaus Hartke, Michael van Hartskamp, Tanguy Ropitault, Jürgen Schönwälder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

Contributors

Ivaylo Petrov
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne
France

Email: ivaylo@ackl.io

Authors' Addresses

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

Email: michel.veillette@trilliant.com

Peter van der Stok (editor)
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: stokcons@bbhmail.nl
URI: www.vanderstok.org

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne
France

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
United States of America

Email: andy@yumaworks.com

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org