Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: May 4, 2016

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
S. Somaraju
Tridonic GmbH & Co KG
R. Somaraju
Landis+Gyr
November 1, 2015

Constrained Objects Language
draft-veillette-core-cool-00

Abstract

This document describes a management interface adapted to constrained devices and constrained (e.g., low-power, lossy) networks. CoOL resources (datastores, protocol operations and notifications) are defined using the YANG modelling language [RFC6020]. Interactions with these resources are performed using the CoAP web transfer protocol [RFC7252]. Payloads and specific CoAP options are encoded using the CBOR data format [RFC7049].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

CoOL is based on the current [I-D.vanderstok-core-comi] draft but instead of using YANG hashes to identify objects, CoOL uses structured identifiers. This approach reduces both message size and implementation footprints. This approach facilitates its use in both centralized (e.g. Network Management System, Path Computation Element), and decentralized scenarios.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification makes use of the following terminology:

- CoOL client: The originating endpoint of a request, and the destination endpoint of a response.

- CoOL server: The destination endpoint of a request, and the originating endpoint of a response.

- Data node: A node in the YANG schema that can be instantiated in a Datastore. One of container, leaf, leaf-list, list, or anyxml.

- Data node ID (DNID): Identifier automatically or manually assigned to a data node. This identifier is unique within the scope of a YANG module.

- Data tree: Hierarchy of instantiated data nodes.

- Child data node: A data node defined within a container or a list is a child of this container or list. The container or list is the parent of the data node.

- Datastore resource: Resource used to store and access information.

- Endpoint: An entity participating in the CoOL protocol. Multiple CoOL endpoints may be accessible using a single CoAP endpoint. In this case, each CoOL enpoint is accessed using a distinct URI.

- Event stream resource: Resource used to access event notifications generated by a CoOL server. Events are defined using the YANG notification statement.

- Fully qualified data node ID (FQDNID): Concatenation of the Module ID and the Data node ID. This identifier uniquely identifies a data node.

- Identifier: An identifier embodies the information required to distinguish what is being identified from all other things within its scope of identification.

- Instance selector: A CBOR array containing the information required to identify one or multiple entries within a YANG list.

- Module ID: Registered identifier assigned to a YANG module.

- Notification ID: Identifier automatically or manually assigned to a YANG notification. This identifier is unique within the scope of a YANG module.

- Object: Within CoOL, objects are a data node within a datastore resource, an RPC within a protocol operation resource, or a notification within an event stream resource.

- Parent data node: See Child data node.

- Protocol operation resource: Resource used to invoke remote procedure calls as defined by the YANG "rpc" statement.

- Protocol operation ID: Identifier automatically or manually assigned to a YANG "rpc". This identifier is unique within the scope of a YANG module.

- Resource: Content identified by a URI.

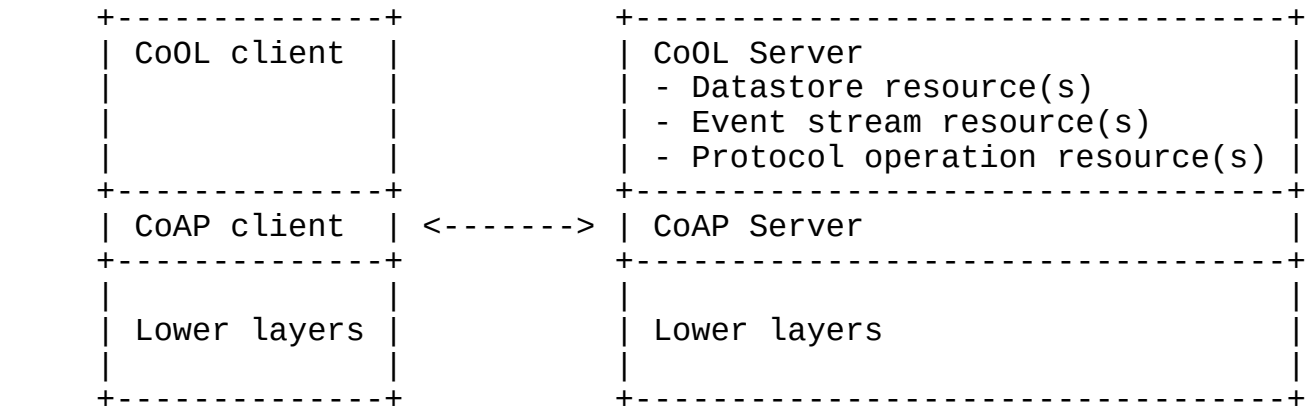
- Schema tree: Hierarchy of data nodes specified within a module.

2. Architecture

The CoOL protocol is based on the client-server model. The CoOL server is the provider of the datastore resource, the protocol operation resource, and the notification resource. The CoOL client is the requester of these resources.

CoOL objects are defined using the YANG modeling language [RFC6020]. Interactions with these objects are performed using the Constrained Application Protocol (CoAP) [RFC7252]. Payloads are encoded using the Concise Binary Object Representation (CBOR) [RFC7049].

This specification is applicable to any transport or security protocols supported by CoAP. Implementers are free to select the most appropriate transport for the targeted applications.



3. CoAP Interface

This section lists the URIs recommended for the different CoOL resources. A CoOL server MAY implement a different set of URIs. (See the Resource discovery section (Section 7.15) for more details on how a CoOL client can discover the list of URIs supported by a CoOL server using the "/.well-known/core" resource.)

- /cool - URI used to access the datastore resource of the default endpoint.
- /cool/rpc - URI used to access the protocol operation resource of the default endpoint.
- /cool/ep0, /cool/ep1, ... - URI used to access the datastore resource of a specific endpoint.
- /cool/ep0/rpc, /cool/ep1/rpc, ... - URI used to access the protocol operation resource of a specific endpoint.
- /cool/stream - URI used to access the default event stream for this device.
- /cool/ep0/stream, /cool/ep1/stream, ... - URI used to access the default event stream of a specific endpoint.
- /cool/stream/0, /cool/stream/1, ... - URI used to access alternate event streams.

Support of endpoints and event streams are optional. The CoAP response code 4.04 (Not Found) MUST be returned when a CoOL client tries to access a resource that is unavailable.

4. Textual representation of CBOR contents

CoOL encodes payloads and the "Fields" CoAP option using the Concise Binary Object Representation (CBOR) as defined by [RFC7049]. Within this document, this binary encoding is represented using an equivalent textual form. This textual form is used strictly for documentation purposes and is never transmitted as such.

The following table summarizes this representation. To facilitate its understanding, this representation follows the JSON syntax (when possible).

CBOR content	CBOR type	Text representation	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 name/value pair 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

5. YANG to CBOR mapping

Objects defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC6020] and CBOR [RFC7049].

5.1. YANG leaf

The leaf statement defines a data node associated with a value. The following subsections describe the encoding of different leaf types.

5.1.1. YANG type: binary

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 0).

Definition example:

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

Textual form: h'1f1ce6a3f42660d888d92a4d8030476e'

CBOR encoding: 50 1f1ce6a3f42660d888d92a4d8030476e

5.1.2. YANG type: bits

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 0). Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are assigned similarly. Within each byte, bits are assigned from least to most significant.

Definition example [RFC6020]:

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

Textual form: h'05' (Represents bits disable-nagle and 10-Mb-only set)

CBOR encoding: 41 05

5.1.3. YANG type: boolean

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

Definition example [RFC7317]:

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

Textual form: true

CBOR encoding: f5

5.1.4. YANG type: decimal64

Leafs of type decimal64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

Definition example [RFC7317]:

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

Textual form: 257 (Represents decimal value 2.57)

CBOR encoding: 19 0101

5.1.5. YANG type: empty

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

Definition example [RFC7277]:

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

Textual form: null

CBOR encoding: f6

5.1.6. YANG type: enumeration

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer data item (major type 0).

Definition example [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; }
  }
}
```

Textual form: 3 (Represents enumeration value "testing")

CBOR encoding: 03

5.1.7. YANG type: identityref

Leafs of type identityref MUST be encoded using a CBOR text string data item (major type 3). Unlike XML, CBOR does not support namespaces. To overcome this limitation, identities are encoded using a concatenation of the identity name(s) of the referenced identities, excluding the base identity and separated by dot(s).

Definition example [RFC7223]:

```
identity interface-type {
}

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

identity ethernetCsmacd {
  base iana-interface-type;
}

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

Textual form: "iana-interface-type.ethernetCsmacd"

CBOR encoding: 78 22
69616e612d696e746572666163652d747970652e6574686572e657443736d616364

5.1.8. YANG type: instance-identifier

When a leaf node of type instance-identifier identifies a single instance data node (data node not part of a list), its value MUST be encoded using a CBOR unsigned integer data item (major type 0) containing the targeted data node ID.

Definition example [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

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

Textual form: 69635

CBOR encoding: 1a 00011003

In this example, the value 69635 identifies the instance of the data node "hostname" within the ietf-system module. Assuming module ID = 68 and data node ID = 3.

When a leaf node of type instance-identifier identifies a data node supporting multiple instances (data node part of a list), its value MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- a CBOR unsigned integer data item (major type 0) containing the fully-qualified data node ID of the targeted data node.

- a CBOR array data item (major type 4) containing the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the "key" YANG statement, starting from top level list, and follow by each of the subordinate list(s).

Definition example [RFC7317]:

```
list user {
  key name;

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

  list authorized-key {
    key name;

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

Textual form: [69679, ["bob", "admin"]]

CBOR encoding: 82 1a 0001102f 82 63 626f62 65 61646d696e

This example identifies the instance of the data node "key-data" within the ietf-system module, associated with user name "bob" and authorized-key name "admin". Assuming module ID = 68 and data node ID = 47.

5.1.9. YANG type: int8, int16, int32, int64

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

Definition example [RFC7317]:

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

Textual form: -300

CBOR encoding: 39 012b

5.1.10. YANG type: leafref

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

Definition example [RFC7223]:

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

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

Textual form: "eth1.10"

CBOR encoding: 67 657468312e3130

5.1.11. YANG type: string

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

Definition example [RFC7223]:

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

Textual form: "eth0"

CBOR encoding: 64 65746830

5.1.12. YANG type: uint8, uint16, uint32, uint64

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

Definition example [RFC7277]:

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

Textual form: 1280

CBOR encoding: 19 0500

5.1.13. YANG type: union

Leafs of type union MUST be encoded using the rules associated with one of the type listed.

Definition example [RFC7317]:

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

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

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

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

Textual form: "[2001:db8:0:1]:80"

CBOR encoding: 71 5b323030313a6462383a303a315d3a3830

5.1.14. YANG type: anyxml

The "anyxml" statement represents an unknown data node. The encoding of this data node MUST follow the rules of one of the YANG statements listed in Section 5.

Definition example [RFC6020]:

```
anyxml data;
```

Textual form: 123

CBOR encoding: 18 7b

Alternate value:

```
{
  1 : 2,
  2 : 55
}
```

CBOR encoding: a2 01 02 02 18 37

5.1.15. YANG type: container

A container MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. CBOR map keys MUST be encoded using a CBOR unsigned integer (major type 0) and set to a data node ID or a fully-qualified data node ID. Data node IDs MUST be used when a parent node exists and this parent shares the same module ID as the current data node. CBOR map values MUST be encoded using the rules associated with the data node type.

Definition example [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 clock {
  leaf current-datetime {
    type date-and-time;
  }

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

Textual form:

```
{
  69667 : {
    36 : "2015-10-02T14:47:24Z-05:00",
    37 : "2015-09-15T09:12:58Z-05:00"
  }
}
```

CBOR encoding:

```
a1
  1a 00011023
  a2
    18 24
    78 1a 323031352d31302d30325431343a34373a32345a2d30353a3030
    18 25
    78 1a 323031352d30392d31355430393a31323a35385a2d30353a3030
```

In this example, we assume that the module ID = 68, data node IDs clock = 35, current-datetime = 36 and boot-datetime 37.

5.1.16. YANG type: leaf-list

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry MUST be encoded using the rules defined by the type specified.

Definition example [RFC7317]:

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

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

Textual form: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662e6f7267 68 696565652e6f7267

5.1.17. YANG type: list

A list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array is encoded using a CBOR map data item (major type 5) following the same rules as a YANG container.

Definition example [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;
  }
}
```

Textual form:

```
{
  69642 : [
    {
      11: "NRC TIC server",
      12 : {
        13: "tic.nrc.ca",
        14: 123
      },
      15 : 0,
      16 : false,
      17 : true
    },
    {
      11: "NRC TAC server",
      12 : {
        13: "tac.nrc.ca"
      }
    }
  ]
}
```

CBOR encoding:

```
a1
   1a 0001100a
   82
      a5
         0b 6e 4e524320544943207365727665 72
         0c a2
            0d 6a 7469632e6e72632e6361
            0e 18 7b
         0f 00
         10 f4
         11 f5
      a2
         0b 6f 4e524320544143207365727665 7220
         0c a1
            0d 6a 7461632e6e72632e6361
```

In this example, we assume that the module ID = 68, data node IDs server = 10, name = 11, udp = 12, address = 13, port = 14, association-type = 15, iburst = 16, prefer = 17.

5.1.18. YANG type: choice

YANG allows the data model to segregate incompatible nodes into distinct choices using the "choice" and "case" statements. Encoded payload MUST carry data nodes defined in only one of the possible cases.

Definition example [RFC7317]:

```
typedef timezone-name {
  type string;
}

choice timezone {
  case timezone-name {
    leaf timezone-name {
      type timezone-name;
    }
  }
  case timezone-utc-offset {
    leaf timezone-utc-offset {
      type int16 {
        range "-1500 .. 1500";
      }
      units "minutes";
    }
  }
}
```

Textual form:

```
{
  69638 : "Europe/Stockholm"
}
```

CBOR encoding:

```
a1
   1a 00011006
   70
      4575726f70652f53746f636b686f6c6d
```

6. Identifiers

All CoOL objects are associated with a unique identifier. The uniqueness of these identifiers is guaranteed by the registration of a module ID used as a prefix for the different manually or automatically assigned identifiers.

6.1. Module ID

A module ID is shared by all objects defined by a module. This includes data nodes, protocol operations, and notifications defined by a YANG module and included YANG sub-modules. Module IDs SHALL be registered; see Section 10.1 for more details.

A registered module ID can be added to a YANG definition file using the YANG extension module-id. Refer to Appendix A for the definition of this extension.

Example:

```
module example {

  import ietf-cool {
    prefix "cool";
  }

  cool:module-id "123";

  ...
}
```

6.2. Data node ID (DNID)

Each data node defined within a YANG module and included YANG sub-module(s) MUST be associated with a unique identifier within the scope of this module.

Data node IDs can be assigned automatically or manually. Data node IDs are assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned data node followed by the associated identifier; these two parts are separated by a space.

Example:

```
cool:id "/container-name/leaf-name 5";
```

When assigned automatically, data nodes are numbered based on their location in the schema tree.

- Top level data nodes are listed in the same order as defined in the YANG module or sub-module.

- Top level data nodes defined in sub-modules are listed after those defined in the module. Sub-modules are ordered based on the order of the include statements in the associated module.

- Data nodes within containers or lists are listed following their parent node and in the same order as defined.

- Data nodes defined within an augment YANG statement are considered top-level nodes and numbered accordingly.

- The "first-assigned-node-id" YANG extension can be used to control the data node ID of the first data node automatically assigned. When this extension is absent, the first data node ID is one. Subsequent data nodes are assigned sequentially.

- Data node that are manually assigned are skipped.

- It is the responsibility of the module author to avoid any conflicts between the manually and automatically assigned data node IDs. Manually assigned identifiers MUST be either lower than the value of "first-assigned-node-id" extension or higher than any identifiers assigned automatically.

- When a module is updated, old Data node IDs can be preserved by adding top level data nodes to the end of the module or by manually assigning new data nodes within the schema tree.

- Data node ID zero is reserved for a virtual root containing all top level objects for a module.

6.3. Fully-qualified data node ID (FQDNID)

The fully-qualified data node ID is the concatenation of the module ID and the data node ID. A fully-qualified data node ID is globally unique.

The fully-qualified data node ID is constructed as follows:

- Bits 0 to 9 (least significant bits) are set to the data node ID assigned to the data node.

- Bits 10 to 29 are set to the module ID registered to the YANG module containing the data node.

- Bits 30 to 31 are reserved and set to zero.

Data node IDs MUST be used when the module ID can be inferred from the parent data node in a YANG container or YANG list, or from the

previous ID in the array of the "Fields" CoAP option. Otherwise, the fully qualified ID MUST be used.

6.4. Notification ID

Each notification defined MUST be associated with a unique identifier within the scope of its associated YANG module and YANG sub-module(s). Notification IDs can be assigned automatically or manually.

Notification IDs can be assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned notification followed by the associated identifier.

Example:

```
cool:id "/system-status-update 1";
```

When assigned automatically, Notification IDs are assigned as follows:

- Notifications are ordered based on their location in the YANG definition file.
- Notifications defined in the sub-module are listed after those defined in the module. Sub-modules are ordered based on the order of the include statements in the associated module.
- The "first-assigned-notification-id" YANG extension can be used to control the notification ID of the first notification automatically assigned. When this extension is absent, the first notification ID is one. Subsequent notifications are assigned sequentially.

6.5. Notification parameter ID

Each notification parameter MUST be associated with a unique identifier within the scope of this notification.

Notification parameter IDs can be assigned automatically or manually. Notification parameter IDs are assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned notification parameter followed by the associated identifier.

Example:

```
cool:id "/system-status-update/current-time 2";
```

When assigned automatically, Notification parameter IDs are assigned as follows:

- Top level parameters are listed in the same order as defined.
- Parameters defined within a container or a list are listed following their parent node and in the same order as defined.
- The first parameter is assigned to ID one, subsequent parameters are assigned sequentially.
- Manually assigned parameters are skipped.

6.6. Protocol operation ID

Each protocol operation defined MUST be associated with a unique identifier within the scope of its associated YANG module and YANG sub-module(s).

Protocol operation IDs can be assigned automatically or manually. Protocol operation IDs are assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned protocol operation followed by the associated identifier.

Example:

```
cool:id "/system-shutdown 3";
```

When assigned automatically, Protocol operation IDs are assigned as follows:

- RPCs are ordered based on their location in the YANG definition file.
- RPCs defined in the sub-module are listed after those defined in the module. Sub-modules are ordered based on the order of the include statements in the associated module.
- The "first-assigned-rpc-id" YANG extension can be used to control the protocol operation ID of the first RPC automatically assigned. When this extension is absent, the first protocol operation ID is one. Subsequent RPCs are assigned sequentially.

6.7. Input parameter ID

Each input parameter MUST be associated with a unique identifier within the scope of the RPC input.

Input parameter IDs can be assigned automatically or manually. Input parameter IDs are assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned input parameter followed by the associated identifier.

Example:

cool:id "/is-supported/input/module-id 1";

When assigned automatically, Input parameter IDs are assigned as follows:

- Top level input parameters are listed in the same order as defined.
- Input parameters defined within a container or a list are listed following their parent nodes and in the same order as defined.
- The first input parameter is assigned to ID one, subsequent input parameters are assigned sequentially.
- Manually assigned input parameters are skipped.

6.8. Output parameter ID

Each output parameter MUST be associated with a unique identifier within the scope of the RPC output.

Output parameter IDs can be assigned automatically or manually. Output parameter IDs are assigned manually using the YANG extension "id". The argument of this extension contains the path of the assigned output parameter followed by the associated identifier.

Example:

cool:id " /is-supported/output/supported 1";

When assigned automatically, Output parameter IDs are assigned as follows:

- Top level output parameters are listed in the same order as defined.
- Output parameters defined within a container or a list are listed following their parent node and in the same order as defined.
- The first output parameter is assigned to ID one, subsequent output parameters are assigned sequentially.

o Manually assigned output parameters are skipped.

6.9. Identifier examples

YANG modules do not need to be updated to be implemented using CoOL. Identifiers can be automatically assigned without the presence of any YANG extensions. Following is the outcome of this process applied to the ietf-system YANG module [RFC7317].

DNID	FQDNID	Data node
0	69632	/
1	69633	/system
2	69634	/system/contact
3	69635	/system/hostname
4	69636	/system/location
5	69637	/system/clock
6	69638	/system/clock/timezone/timezone-name/ timezone-name
7	69639	/system/clock/timezone/timezone-utc-offset /timezone-utc-offset
8	69640	/system/ntp
9	69641	/system/ntp/enabled
10	69642	/system/ntp/server
11	69643	/system/ntp/server/name
12	69644	/system/ntp/server/transport/udp/udp
13	69645	/system/ntp/server/transport/udp/udp/address
14	69646	/system/ntp/server/transport/udp/udp/port
15	69647	/system/ntp/server/association-type
16	69648	/system/ntp/server/iburst
17	69649	/system/ntp/server/prefer
18	69650	/system/dns-resolver
19	69651	/system/dns-resolver/search
20	69652	/system/dns-resolver/server
21	69653	/system/dns-resolver/server/name
22	69654	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp
23	69655	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp/address
24	69656	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp/port
25	69657	/system/dns-resolver/options
26	69658	/system/dns-resolver/options/timeout
27	69659	/system/dns-resolver/options/attempts
28	69660	/system/radius
29	69661	/system/radius/server
30	69662	/system/radius/server/name

31	69663	/system/radius/server/transport/udp/udp
32	69664	/system/radius/server/transport/udp/udp/address
33	69665	/system/radius/server/transport/udp/udp/ authentication-port
34	69666	/system/radius/server/transport/udp/udp/ shared-secret
35	69667	/system/radius/server/authentication-type
36	69668	/system/radius/options
37	69669	/system/radius/options/timeout
38	69670	/system/radius/options/attempts
39	69671	/system/authentication
40	69672	/system/authentication/user-authentication-order
41	69673	/system/authentication/user
42	69674	/system/authentication/user/name
43	69675	/system/authentication/user/password
44	69676	/system/authentication/user/authorized-key
45	69677	/system/authentication/user/authorized-key/name
46	69678	/system/authentication/user/authorized-key/ algorithm
47	69679	/system/authentication/user/authorized-key/ key-data
48	69680	/system-state
49	69681	/system-state/platform
50	69682	/system-state/platform/os-name
51	69683	/system-state/platform/os-release
52	69684	/system-state/platform/os-version
53	69685	/system-state/platform/machine
54	69686	/system-state/clock
55	69687	/system-state/clock/current-datetime
56	69688	/system-state/clock/boot-datetime

Protocol operation IDs	Protocol operation	
1	/set-current-datetime	
	Input parameter ID	Input parameter
	1	/current-datetime
2	/system-restart	
3	/system-shutdown	

If we assume that the extensions shown below have been added to this module:

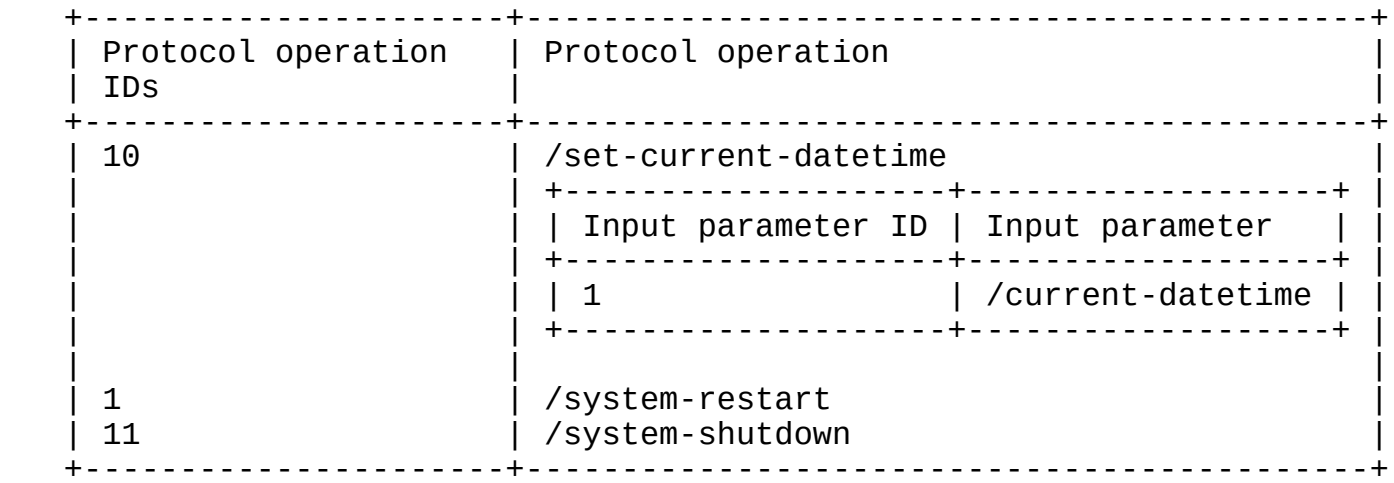
Example:

```
module ietf-system {

  import ietf-cool {
    prefix "cool";
  }

  cool:module-id "68";
  cool:first-assigned-node-id "20"
  cool:first-assigned-rpc-id "10"
  cool:id "/system/location 1"
  cool:id "/system/dns-resolver/server/name 2"
  cool:id "/system-restart 1"

  ...
}
```

The generated identifiers will be affected as follows:

DNID	FQDNID	Data node
0	69632	/
20	69652	/system
21	69653	/system/contact
22	69654	/system/hostname
1	69633	/system/location
23	69655	/system/clock
24	69656	/system/clock/timezone/timezone-name/ timezone-name
25	69657	/system/clock/timezone/timezone-utc-offset /timezone-utc-offset
26	69658	/system/ntp
27	69659	/system/ntp/enabled
28	69660	/system/ntp/server
29	69661	/system/ntp/server/name
30	69662	/system/ntp/server/transport/udp/udp
31	69663	/system/ntp/server/transport/udp/udp/address
32	69664	/system/ntp/server/transport/udp/udp/port
33	69665	/system/ntp/server/association-type
34	69666	/system/ntp/server/iburst
35	69667	/system/ntp/server/prefer
36	69668	/system/dns-resolver
37	69669	/system/dns-resolver/search

38	69670	/system/dns-resolver/server
2	69634	/system/dns-resolver/server/name
39	69671	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp
40	69672	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp/address
41	69673	/system/dns-resolver/server/transport/ udp-and-tcp/udp-and-tcp/port
42	69674	/system/dns-resolver/options
43	69675	/system/dns-resolver/options/timeout
44	69676	/system/dns-resolver/options/attempts
45	69677	/system/radius
46	69678	/system/radius/server
47	69679	/system/radius/server/name
48	69680	/system/radius/server/transport/udp/udp
49	69681	/system/radius/server/transport/udp/udp/address
50	69682	/system/radius/server/transport/udp/udp/ authentication-port
51	69683	/system/radius/server/transport/udp/udp/ shared-secret
52	69684	/system/radius/server/authentication-type
53	69685	/system/radius/options
54	69686	/system/radius/options/timeout
55	69687	/system/radius/options/attempts
56	69688	/system/authentication
57	69689	/system/authentication/user-authentication-order
58	69690	/system/authentication/user
59	69691	/system/authentication/user/name
60	69692	/system/authentication/user/password
61	69693	/system/authentication/user/authorized-key
62	69694	/system/authentication/user/authorized-key/name
63	69695	/system/authentication/user/authorized-key/ algorithm
64	69696	/system/authentication/user/authorized-key/ key-data
65	69697	/system-state
66	69698	/system-state/platform
67	69699	/system-state/platform/os-name
68	69700	/system-state/platform/os-release
69	69701	/system-state/platform/os-version
70	69702	/system-state/platform/machine
71	69703	/system-state/clock
72	69704	/system-state/clock/current-datetime
73	69705	/system-state/clock/boot-datetime



7. Protocol details

This section defines the different interactions supported between a CoOL client and a CoOL server.

7.1. Retrieving data node(s)

The CoAP GET method is used by a CoOL client to retrieve the value of one or multiple data nodes.

The URI of the GET request MUST be set to the URI of the targeted datastore.

The data node(s) to be retrieved MUST be specified within the "Fields" CoAP option. This CoAP option contains a list of data node IDs encoded using a CBOR array. The first data node ID MUST be fully-qualified. Subsequence IDs MUST be elided if the module ID is shared with the previous array entry.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content). The payload of the GET response MUST carry a CBOR map containing the data node(s) requested. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map. The CBOR value undefined (0xf7) must be returned for each data node requested but not currently available.

Example:

CoAP request:
GET /cool Fields([69637, 55])

CoAP response:
2.05 Content Content-Format(application/cbor)

```
{
  69637 : {
    7 : 540
  },
  69687 : "2015-10-08T14:10:08Z09:00"
}
```

In this example, a CoOL client retrieves the data nodes "/system/clock" and "/system-state/clock/current-datetime" defined in the ietf-system module.

This example makes use of the following IDs:

- module ietf-system: module ID 68
- leaf clock: data node ID 5, fully-qualified ID 69637
- leaf timezone-utc-offset: data node ID 7
- leaf current-datetime: data node ID 55, fully-qualified ID 69687

These CoAP requests and responses MUST be encoded in accordance with [RFC7252]. An encoding example is shown below:

CoAP request:

CoAP field	Size (bytes)	Value
Version, Type, Token Length, Code	2	
Message ID	2	
Token ID	0 to 8	
option Uri-Path (Delta and Length)	1	
option Uri-Path (Value)	5	"/cool"
option Fields (Delta and Length)	1	
option Fields (Value)	8	82 1a 00011005 18 37

CoAP response:

CoAP field	Size (bytes)	Value
Version, Type, Token Length, Code	2	
Message ID	2	
Token ID	0 to 8	
Option Content-Format (Delta and Length)	1	
Option Content-Format (Value)	1	60
Payload	43	a2 1a 00011005 a1 07 19 021c 1a 00011037 78 19 32303135 2d31302d 30385431 343a3130 2d30385a 30393a30 30

7.2. The Fields CoAP option

The "Fields" CoAP option is used to specify the list of data node(s) accessed. This option contains a CBOR array. Each entry of this array can be an unsigned integer representing a data node ID or a CBOR array representing an instance selector.

Example:

No.	C	U	N	R	Name	Format	Length	Default
6	x	x	-	x	Fields	opaque	0-3 B	(none)

7.3. Retrieving all data nodes

To retrieve all data nodes associated with a YANG module, the Fields CoAP option MUST be present and the CBOR array MUST contain the data node ID zero for this module. Data nodes added to the specified module by other modules using the augment YANG statement are returned

but data nodes added by the specified module to other modules are not returned.

To retrieve all data nodes of all YANG modules, the Fields CoAP option MUST be absent.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content). The payload of the GET response MUST carry a CBOR map containing a container for each module reported and identified using the data node ID zero.

Example:

CoAP Request:

```
GET /cool Fields([66560])
```

CoAP response:

```
2.05 Content Content-Format(application/cbor)
{
  66560 : {
    1 : {
      2 : [
        {
          3 : "eth0",
          4 : "Ethernet adapter Local Area Connection",
          5 : "iana-interface-type.ethernetCsmacd",
          6 : true,
          68620 : {
            13 : true,
            14 : true,
            15 : 1280,
            16 : [
              {
                17 : "fe80::200:f8ff:fe21:67cf",
                18 : 10
              }
            ]
          }
        }
      ]
    }
  }
}
```

In this example, a CoOL client retrieves all data nodes of the YANG module "ietf-interfaces". Data nodes defined using the augment YANG statement in the "ietf-ip" module are also returned.

If we assume that the CoOL server implements only the "ietf-interfaces" and the "ietf-ip" modules, then the following request returns the same response as above.

```
GET /cool
```

This example makes use of the following IDs:

- module ietf-interfaces: module ID 65
- module root : data node ID 0, fully-qualified 66560
- container interfaces: data node ID 1
- list interface: data node ID 2
- leaf name: data node ID 3
- leaf description: data node ID 4
- leaf type: data node ID 5
- leaf enabled: data node ID 6
- module ietf-ip: module ID 67
- container ipv6: data node ID 12, fully-qualified 68620
- leaf enabled: data node ID 13
- leaf forwarding: data node ID 14
- leaf mtu: data node ID 15
- list address: data node ID 16
- leaf ip: data node ID 17
- leaf prefix-length: data node ID 18

7.4. Updating data node(s)

The CoAP PUT method is used by CoOL clients to change the value of one or multiple data nodes.

The URI of the PUT request MUST be set to the URI of the targeted datastore.

The payload of the PUT request carry a CBOR map containing the list of data node(s) to be updated. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map. It is important to note that the CBOR map itself does not represent a data node; rather, each tag-value pair in that map represents a data node to be updated.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.04 (Changed). If at least one of the targeted data nodes doesn't exist, the CoOL server MUST return a response code 4.00 (Bad Request) carrying a payload with the data node "/error-info/error-code" set to 4 (doesNotExist). PUT methods SHOULD be processed as atomic transactions, if any errors occur, then the target datastore SHOULD NOT be changed by the PUT operation.

Example:

```
CoAP request:
  PUT /cool Content-Format(application/cbor)
  {
    69639 : 540,
    69687 : "2015-10-08T14:10:08Z09:00"
  }

CoAP response:
  2.04 Changed
```

In this example, a CoOL client updates data nodes /system/clock/timezone/timezone-utc-offset/timezone-utc-offset" and "/system-state/clock/current-datetime".

This example makes use of the following IDs:

- module ietf-interfaces: module ID 65
- leaf timezone-utc-offset: data node ID 7, fully-qualified 69639
- leaf current-datetime: data node ID 55, fully-qualified 69687

7.5. Retrieving data node(s) from a list

The CoAP GET method is used by CoOL clients to retrieve the value of one or multiple data nodes within a list.

To retrieve data node(s) within a list, the Fields option MUST carry an instance selector. An instance selector is a CBOR array containing these elements:

- The first element of the CBOR array MUST be set to the data node ID of the list containing the requested data nodes.

- The second element of the array MUST be set to a CBOR array containing the value of the different keys required to select the requested data nodes. Key values MUST be provided in the same order as defined in the YANG key sub-statement. When a list is defined within list(s), the key values are ordered starting with the top level list and ending with the list containing the requested data nodes.

- The third element is optional. When present, this element MUST be set to a CBOR array containing the data node IDs of the data nodes requested. When absent, all data nodes within the targeted list entry MUST be returned.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content). The payload of the GET response MUST carry a CBOR map containing the requested data nodes. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

When the list entry specified in the request does not exist, the returned payload MUST contain a map entry set to the CBOR value undefined (0xf7).

Example:

```
CoAP request:
  GET /cool Fields([69635, [66562, ["eth0"], [5, 6]])

CoAP response:
  2.05 Content Content-Format(application/cbor)
  {
    69635 : "datatracker.ietf.org",
    66562 : {
      5 : "iana-interface-type.ethernetCsmacd",
      6 : true
    }
  }
```

In this example, a CoOL client retrieves the data node "/system/hostname" defined in the ietf-system module and the data nodes "/interfaces/interface/type" and "/interfaces/interface/enabled" member of the "/interfaces/interface" list defined in the ietf-interfaces module.

```
Alternname CoAP response:
  2.05 Content Content-Format(application/cbor)
  {
    69635 : "datatracker.ietf.org",
    66562 : undefined
  }
```

This alternate CoAP response represents the case where the interface name "eth0" doesn’t exist within the "/interfaces/interface" list.

This example makes use of the following IDs:

- module ietf-interfaces: module ID 65
- module ietf-system: module ID 68
- leaf hostname: data node ID 3, fully-qualified 69635
- list interface: data node ID 2, fully-qualified 66562
- leaf type: data node ID 5
- leaf enabled: data node ID 6

7.6. Retrieving multiple entries from a list

Two approaches are supported to retrieve multiple list entries.

The first approach consists of providing a list of key sets. Each key set selects one entry within the YANG list. In this case, the second element of the instance selector MUST be encoded as a CBOR array for which each entry is itself a CBOR array containing the key(s).

The second approach consists of setting one or multiple keys to null (0xf6). In this case, all possible values for this key are returned.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content). The payload of the GET response MUST carry a CBOR map containing the different data nodes requested. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

When the list entry specified in the request does not exist, the returned payload MUST contain a map entry set to the CBOR value undefined (0xf7).

Example:

CoAP request:
GET /cool Fields([[66569,[["eth0"], ["eth1"]], [10, 22, 27]]])

CoAP response:
2.05 Content Content-Format(application/cbor)

```
{
  66569 : [
    {
      10 : "eth0",
      22 : 1572864,
      27 : 0
    },
    {
      10 : "eth1",
      22 : 1572864,
      27 : 0
    }
  ]
}
```

In this example, a CoOL client retrieves data nodes "/interfaces-state/interface/statistics/in-octets" and "/interfaces-state/interface/statistics/in-errors" for the interface name "eth0" and "eth1".

Assuming that only two interfaces exist for this host, the following request returns the same response as above.

CoAP request:
GET /cool Fields([[66569,[null], [10, 22, 27]]])

This example makes use of the following IDs:

- module ietf-interfaces: module ID 65
- list interface: data node ID 9, fully-qualified 66569
- leaf name: data node ID 10
- leaf in-octets: data node ID 22
- leaf in-errors: data node ID 27

7.7. Retrieving multiple entries of a single data node from a list

Retrieval of a single data node within a list MUST be optimized as follows:

In the request, the third element of the instance selector MUST be encoded as a CBOR unsigned integer instead of a CBOR ARRAY.

In the response, the instances of the selected data node MUST be encoded as a CBOR ARRAY associated directly with the ID of this data node.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.05 (Content). When the list entry specified in the request does not exist, the returned payload MUST contain a map entry set to the CBOR value undefined (0xf7).

Example:

CoAP request:
```
GET /cool Fields([ [66569,[ null ], 10] ])
```

CoAP response:
```
2.05 Content Content-Format(application/cbor)
{
  66570 : [ "eth0", "eth1", "wlan0" ]
}
```

In this example, a CoOL client retrieves all instances of data node "/interfaces-state/interface/name" within the "/interfaces-state/interface" list.

This example makes use of the following IDs:

- module ietf-interfaces: module ID 65
- list interface: data node ID 9, fully-qualified 66569
- leaf name: data node ID 10, fully-qualified 66570

7.8. Updating a list entry

The CoAP PUT method is used by a CoOL client to update the value of one or multiple data nodes within a list.

The URI of the PUT request MUST be set to the URI of the targeted datastore.

The Fields CoAP option MUST be set to an instance selector composed of the following elements:

- The first element MUST be set to the data node ID of the list.
- The second element MUST be set to a CBOR array containing the value of the different keys required to select the targeted entry within the specified list.

The payload of the PUT request MUST carry a CBOR map containing the list entry to be updated. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.04 (Changed). If the list specified doesn't exist, the CoOL server MUST return a response code 4.00 (Bad Request) carrying a payload with data node "/error-info/error-code" set to 4 (doesNotExist). PUT methods SHOULD be processed as atomic transactions, if any errors occur, then the target datastore SHOULD NOT be changed by the PUT operation.

Example:

```
CoAP request:
  PUT /cool Fields([69642,[ "NTP Pool server 2" ]])
  {
    69642 : {
      12 : {
        13: "2620:10a:800f::11",
        14: 123
      }
    }
  }

CoAP response:
  2.04 Changed
```

In this example, a CoOL client update data nodes "/system/ntp/server/transport/udp/udp/address" and "/system/ntp/server/transport/udp/udp/port" within the "/system/ntp/server" list.

This example makes use of the following IDs:

- module ietf-system: module ID 68
- list server: data node ID 10, fully-qualified 69642

- container udp: data node ID 12
- leaf address: data node ID 13
- leaf port: data node ID 14

7.9. Adding a list entry

The CoAP POST method is used by a CoOL client to insert an entry into a list.

The URI of the POST request MUST be set to the URI of the targeted datastore.

The payload of the POST request MUST carry a CBOR map containing the data node ID of the list targeted. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.01 (Created). If the entry provided already exists in the targeted list (duplicate key), the CoOL server MUST return a response code 4.00 (Bad Request) carrying a payload with data node "/error-info/error-code" set to 5 (alreadyExist).

Example:

```
CoAP request:
  POST /cool
  {
    69642 : {
      12 : {
        13: "132.246.11.231",
        14: 123
      }
    }
  }

CoAP response:
  2.01 Created
```

In this example, a CoOL client adds an entry in the "/system/ntp/server" list. The entry added contains data nodes "/system/ntp/server/transport/udp/udp/address" and "/system/ntp/server/transport/udp/udp/port".

This example makes use of the following IDs:

- module ietf-system: module ID 68
- list server: data node ID 10, fully-qualified 69642
- container udp: data node ID 12
- leaf address: data node ID 13
- leaf port: data node ID 14

7.10. Deleting list entries

The CoAP DELETE method is used by a CoOL client to delete an entry within a list.

The URI of the PUT request MUST be set to the URI of the targeted datastore.

The Field CoAP option MUST contain an instance selector composed of the following elements:

- The first element MUST be set to the data node ID of the list.
- The second element MUST be set to a CBOR array containing the value of the different keys required to identify the entry to be deleted.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.02 (Deleted). If the targeted entry doesn't exist, the CoOL server MUST return a response code 4.00 (Bad Request) carrying a payload with data node "/error-info/error-code" set to 4 (doesNotExist).

Example:

CoAP request:
```
DELETE /cool Fields([69642,[ "NTP Pool server 2" ]])
```

CoAP response:
```
2.02 Deleted
```

In this example, a CoOL client deletes the entry within the "/system/ntp/server" list for the key "/system/ntp/server/name" = "NTP Pool server 2".

This example makes use of the following IDs:

- module ietf-system: module ID 68

- list server: data node ID 10, fully-qualified 69642

7.11. Patch

The CoAP PATCH method is used by CoOL clients to perform complex atomic transactions on a datastore.

To perform a patch, the CoAP method MUST be set to PATCH.

The payload of the CoAP request MUST carry a "patch-request" list as defined by the ietf-cool module. This list carries a sequence of edits on a datastore. Each edit MUST be applied in ascending order, and all edits MUST be applied. If any errors occur, then the target datastore SHOULD NOT be changed by the patch operation.

On successful processing of the CoAP request, the CoOL server MUST return a CoAP response with a response code 2.04 (Changed).

If at least one of the edit is rejected, the CoOL server MUST return a CoAP response with a response code 4.00 (Bad Request) and the payload MUST carry a "patch-response" list as defined by the ietf-cool module.

In the following example, a CoOL client performs the following operations using the patch method:

- Remove all entries present in the"/system/ntp/server" list.
- Create an entry in the "/system/ntp/server" list with the server name set to "NTP Pool server 2" and the server address set to "2.pool.ntp.org".
- Set the data node "/system/ntp/enabled" to true.

Example:

CoAP request:

```
PATCH /cool Content-Format(application/cbor)
{
  1028 : [
    {
      5 : 0,
      6 : 4,
      7 : [69642, [null]]
    },
    {
      5 : 1,
      6 : 0,
      8 : {
        69642 : {
          11 : "NTP Pool server 2",
          12 : {
            13 : "2.pool.ntp.org"
          }
        }
      }
    },
    {
      5 : 2,
      6 : 3,
      8 : {
        69641 : true
      }
    }
  ]
}
```

CoAP response:

```
2.04 Changed
```

Alternate CoAP response:

```
4.00 Bad Request (Content-Format: application/cbor)
{
  1033 : [
    {
      10 : 1,
      11 : 3
    }
  ]
}
```

This alternate response represents the case where the second edit have been rejected because of an invalid value.

This example makes use of the following IDs:

- module ietf-cool: module ID 1
- list patch-request: data node ID 4, fully-qualified 1028
- leaf operation: data node ID 5
- leaf list-entry: data node ID 6
- anyxml value: data node ID 8
- list patch-response
- leaf edit-id
- leaf edit-status
- module ietf-system: module ID 68
- leaf enabled : data node ID 9, fully-qualified 69641
- list server: data node ID 10, fully-qualified 69642
- leaf name: data node ID 11
- container udp: data node ID 12
- leaf address: data node ID 13

7.12. Protocol operation

Protocol operations are defined using the YANG "rpc" statement. Protocol operations are invoked by CoOL clients using a CoAP POST method on the protocol operation resource.

When input parameter(s) are defined for the invoked protocol operation and are needed by the CoOL client, they are carried in the CoAP request payload. These parameter(s) are encoded using a CBOR map. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

The response code of the CoAP response MUST be set to 2.05 (Content). When output parameters are returned by the CoOL server, these parameter(s) are carried in the CoAP response payload. These

parameter(s) are encoded using a CBOR map. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules used to construct this CBOR map.

Example:

```
CoAP request:
  POST /cool/rpc/68/1 Content-Format(application/cbor)
  {
    1 : "2015-10-08T14:10:08Z09:00"
  }

CoAP response:
  2.05 Content
```

In this example, a CoOL client invokes the protocol operation "/set-current-datetime". The "current-datetime" value is provided as input parameter.

This example makes use of the following IDs:

- module ietf-system: module ID 68
- rpc set-current-datetime: protocol operation ID 1
- input parameter current-datetime: input parameter ID 1

7.13. Event stream

Notifications are defined using the YANG "notification" statement. Subscriptions to an event stream and notification reporting are performed using an event stream resource. When multiple event stream resources are supported, the list of notifications associated with each stream is either pre-defined or configured in the CoOL server. CoOL clients MAY subscribe to one or more event stream resources.

To subscribe to an event stream resource, a CoOL client MUST send a CoAP GET with the Observe CoAP option set to 0. To unsubscribe, a CoOL client MAY send a CoAP reset or a CoAP GET with the Observe option set to 1. For more information on the observe mechanism, see [RFC7641].

Each notification transferred by a CoOL server to each of the registered CoOL client is carried in a CoAP response with a response code set to 2.05 (Content). Each CoAP response MUST carry in its payload at least one notification but MAY carry multiple. Each notification MUST be encoded as a CBOR map. The subsection "YANG container" of the "YANG to CBOR mapping" section defines the rules

used to construct this CBOR map. When multiple notifications are reported, the CoAP response payload carries a CBOR array, with each entry containing a notification encoded using a CBOR map.

For this example, we assume that the following notifications have been added to the ietf-system module.

Example:

```
notification system-status-update {
  leaf status-update {
    type enumeration {
      enum system-down { value 0; }
      enum system-up { value 1; }
    }
  }

  leaf current-time {
    type yang:date-and-time;
  }
}

notification ntp-time-updated {
  leaf new-time {
    type yang:date-and-time;
  }

  leaf time-drift {
    type uint64;
    units "milliseconds";
  }
}
```

In this example, a CoOL client starts by registering to the default event stream resource "/cool/streams".

```
CoAP request:
  GET /cool/streams Observe(0) Token(0x9372)
```

The CoOL server confirms this registration by returning a first CoAP response. The payload of this CoAP response may be empty or may carry the last notification reported by this server.

```
CoAP response:
  2.05 Content Observe(52) Token(0xD937)
```

After detecting an event, the CoOL server sends its first notification to the registered CoOL client.

```
CoAP response:
  2.05 Content Observe(53) Token(0xD937)
               Content-Format(application/cbor)
  {
    69634 : {
      1 : "2015-10-08T14:10:08Z09:00",
      2 : 271
    }
  }
```

To optimize communications or because of other constrains, the CoOL server might transfer multiple notifications in a single CoAP response.

```
CoAP response:
  2.05 Content Observe(52) Token(0xD937)
               Content-Format(application/cbor)
  [
    {
      69633 : {
        1 : 0,
        2 : "2015-10-08T15:23:51Z09:00"
      }
    },
    {
      69633 : {
        1 : 1,
        2 : "2015-10-08T15:25:36Z09:00"
      }
    }
  ]
```

This example makes use of the following IDs:

- module ietf-system: module ID 68
- notification system-status-update: notification ID 69633
- leaf status-update: notification parameter ID 1
- leaf current-time: notification parameter ID 2
- notification ntp-time-updated : notification ID 69634
- leaf updated-time: notification parameter ID 1
- leaf time-drift: notification parameter ID 2

7.14. Observe

A CoOL server MAY support state change notifications to some or all its leaf data nodes. When supported the CoOL server MUST implement the Server-Side requirements defined in [RFC7641] section 3 and the CoOL client MUST implement the Client-Side requirements defined in [RFC7641] section 4.

To start observing a leaf data node, a CoOL client MUST send a CoAP GET with the Observe CoAP option set to 0. The Fields CoAP option may be set to any content previously defined within this section. The first data node selected represents the resource observed as described in [RFC7641]. Subsequent data nodes selected represent coincidental values. These data nodes are included in each notification, but changes to these extra data nodes MUST not trigger notification messages.

A subscription can be terminated by the CoOL client by returning a CoAP Reset message or by sending a GET request with an Observe CoAP option set to deregister (1). More details are available in [RFC7641].

Example:

```
CoAP request:
  GET /cool Fields([ [66594, ["eth0"]], [34, 29]]) Observe(0)

CoAP response:
  2.05 Content Content-Format(application/cbor) Observe(2631)
  {
    66594: {
      29 : 605873,
      34 : 42
    }
  }

  ...

CoAP response:
  2.05 Content Content-Format(application/cbor) Observe(2632)
  {
    66594: {
      29 : 6493721,
      34 : 43
    }
  }
```

In this example, a CoOL client subscribes to state changes of the data node "/interfaces-state/interface/statistics/out-errors" and requests that data node "/interfaces-state/interface/statistics/out-octets" is reported as coincidental value.

A first response is immediately returned by the CoOL server to confirm the subscription and to report the current values of the requested data nodes.

Subsequent responses are returned by the CoOL server each time the data node "/interfaces-state/interface/statistics/out-errors" changes.

7.15. Resource discovery

The "/.well-known/core" resource is used by CoOL clients to discover resources implemented by CoOL servers. Each CoOL server MUST have an entry in the "/.well-known/core" resource for each datastore resource, protocol operation resource, and event stream resource supported.

Resource discovery MUST be performed using a CoAP GET request. The CoAP response MUST have a response code set to 2.05 (Content), a Content-Format set to "application/link-format", and a payload containing a list of web links.

To enable discovery of specific resource types, the CoAP server MUST support the query string "rt".

Link format and the "/.well-known/core" resource are defined in [RFC6690].

Example:

CoAP request:

```
GET /.well-known/core
```

CoAP response:

```
2.05 Content Content-Format(application/link-format)
</cool>;rt="cool.datastore",
</cool/rpc>;rt="cool.protocol-operation",
</cool/stream>;rt="cool.event-stream",
```

In this example, a CoOL client retrieves the list of all resources available on a CoOL server.

Alternatively, the CoOL client may query for a specific resource type. In this example, the CoOL client queries for resource type (rt) "cool.datastore".

CoAP request:
GET /.well-known/core?rt=cool.datastore

CoAP response:
2.05 Content Content-Format(application/link-format)
</cool>;rt="cool.datastore",

7.16. Modules, sub-modules, and objects discovery

CoOL clients can discover the list of modules, sub-modules, data nodes, protocol operations, and event streams implemented by CoOL servers by requesting this information from the "ietf-cool-library" YANG module.

CoOL servers MUST implement the "ietf-cool-library" YANG module. This module MUST contain information about all modules supported by this CoOL server.

The "ietf-cool-library" YANG module also reports detailed information about each module, such as "name", "features", "deviations", "submodule", "data-nodes", "protocol-operation" and "notifications". CoOL servers SHOULD support this detail information if the amount of resources available allow their implementation.

Example:

```
CoAP request:
  GET /cool Fields([2049])

CoAP response:
  2.05 Content Content-Format(application/cbor)
  {
    2049 : [
      {
        2 : 65,
        3 : "ietf-interfaces",
        4 : "2014-05-08",
        5 : ["arbitrary-names", "pre-provisioning"],
        10 : [1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 20, 21,
              22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34]
      },
      {
        2 : 68,
        3 : "ietf-system",
        4 : "2014-08-06",
        5 : ["ntp", "ntp-udp-port"],
        10 : [1, 2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
        11 : [1, 2, 3]
      }
    ]
  }
```

In this example, a CoOL client retrieves the entire content of the "module" list. The CoOL protocol can be used by a CoOL client to retrieve specific information within this list.

Second example:

```
CoAP request:
  POST /cool/rpc/2/1 Content-Format(application/cbor)
  {
    1 : 68,
    2 : 5
  }

CoAP response:
  2.05 Content Content-Format(application/cbor)
  {
    1 : true
  }
```

In this second example, a CoOL client uses the is-supported protocol operation to verify support of data node "/system/clock".

These examples make use of the following IDs:

- module ietf-cool-library: module ID 2
- module ietf-interfaces: module ID 65
- module ietf-system: module ID 68
- list modules: data node ID 1, fully-qualified 2049
- leaf module-id: data node ID 2
- leaf name: data node ID 3
- leaf revision: data node ID 4
- leaf-list features: data node ID 5
- leaf-list data-nodes: data node ID 10
- leaf-list protocol-operations: data node ID 11
- rpc is-supported: protocol operation ID 1

8. Error Handling

All CoAP response codes defined by [RFC7552] MUST be accepted and processed accordingly by CoOL clients. Optionally, client errors (CoAP response codes 4.xx) or server errors (CoAP response codes 5.xx) MAY have a payload providing further information about the cause of the error. This payload contain the "error-info" container defined in the ietf-cool YANG module.

Example:

CoAP response:

```
  4.00 Bad Request (Content-Format: application/cbor)
  {
    1025 : 2,
    1026 : "Data node 69687 is unknown"
  }
```

9. Security Considerations

This application protocol relies on the lower layers to provide confidentiality, integrity, and availability. A typical approach to archive these requirements is to implement CoAP using the DTLS binding as defined in [RFC7252] section 9. Other approaches are possible to fulfill these requirements, such as the use of a network layer security mechanism as discussed in [I-D.bormann-core-ipsec-for-coap] or a link layer security mechanism for exchanges done within a single sub-network (e.g. [I-D.wang-6tisch-6top-coapie].

In some applications, different access rights to CoOL resources and objects need to be granted to different CoOL clients. Different solutions are possible, such as the implementation of Access Control Lists (ACL) using YANG module(s) or the use of an authorization certificate as defined in [RFC5755]. These access control mechanisms need to be addressed in complementary specifications.

The Security Considerations section of CoAP [RFC7252] is especially relevant to this application protocol and should be reviewed carefully by implementers.

10. IANA Considerations

10.1. Module ID

This document defines a registry for YANG module IDs. The name of this registry is "CoOL module ID".

The registry shall record for each entry:

The assigned identifier.

The name of the YANG module.

A reference to the module and associated sub-module documentation (e.g. RFC number).

Contact information of the owner of the module, such as name, email, and phone number.

Module ID	Module name	Reference	Owner
0	Reserved		
1	cool	Defined in annex A	IETF
2	cool-library	Defined in annex B	IETF
3 to 63	Reserved	Expert Review. Reserved for standardized YANG modules targeting constrained networks and devices. These module IDs require less overhead when encoded using CBOR.	
64	ietf-inet-types	[RFC6021]	IETF
65	ietf-interfaces	[RFC7223]	IETF
66	iana-if-type	[RFC7224]	IETF
67	ietf-ip	[RFC7277]	IETF
68	ietf-system	[RFC7317]	IETF
100 to 127	Reserved	Experimental use	
128 to 1023	Reserved	Temporary registry maintained by the authors to allow initial implementations. These identifiers should be transferred to the official registry when appropriate.	Authors

10.2. "Fields" CoAP Option Number

The Fields CoAP option is used to specify a list of data nodes to be retrieved by a CoAP GET. This option needs to be registered in the CoAP Option Numbers registry as defined in [RFC7252] section 12.2.

10.3. "PATCH" CoAP Method Code

This draft makes use of the PATCH CoAP method as defined in [I-D.vanderstok-core-patch]. This method needs to be registered in the CoAP Method Codes sub-registry as defined in [RFC7252] section 12.1.1.

11. References

11.1. Normative References

[I-D.vanderstok-core-patch]
Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.

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

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://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, <http://www.rfc-editor.org/info/rfc7317>.

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

11.2. Informative References

[I-D.bormann-core-ipsec-for-coap]
Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 2012.

[I-D.vanderstok-core-comi]
Stok, P., Bierman, A., Schoenwaelder, J., and A. Sehgal, "CoAP Management Interface", draft-vanderstok-core-comi-08 (work in progress), October 2015.

[I-D.wang-6tisch-6top-coapie]
Wang, Q., Vilajosana, X., Watteyne, T., Sudhaakar, R., and P. Zand, "Transporting CoAP Messages over IEEE802.15.4e Information Elements", draft-wang-6tisch-6top-coapie-01 (work in progress), July 2015.

[RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, DOI 10.17487/RFC5755, January 2010, <http://www.rfc-editor.org/info/rfc5755>.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <http://www.rfc-editor.org/info/rfc6021>.

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

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <http://www.rfc-editor.org/info/rfc7224>.

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

[RFC7552] Asati, R., Pignataro, C., Raza, K., Manral, V., and R. Papneja, "Updates to LDP for IPv6", RFC 7552, DOI 10.17487/RFC7552, June 2015, <http://www.rfc-editor.org/info/rfc7552>.

Appendix A. ietf-cool YANG module

Module containing YANG extensions for the CoOL protocol.

```
module ietf-cool {
  namespace "urn:ietf:ns:cool";
  prefix cool;

  organization
    "IETF Core Working Group";
```

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

        Randy Turner
        <mailto:Randy.Turner@landisgyr.com>

        Alexander Pelov
        <mailto:a@ackl.io>

        Abhinav Somaraju
        <mailto:abhinav.somaraju@tridonic.com>";

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

     revision 2015-09-30 {
       description
         "Initial revision.";
       reference
         "draft-veillette-core-cool";
     }

     // YANG modeling language extensions

     extension module-id {
       description  "YANG statement used to assign a module ID.";
       argument "module-identifier";
     }

     extension first-assigned-node-id {
       description  "YANG statement used to specify the first identifier
                     automatically assigned to a data node within a
                     YANG module. Subsequent data nodes are assigned
                     sequentially. Values lower than the
                     first-assigned-node-id are reserved for
                     manual assignment.";
       argument "first-data-node-identifier";
     }

     extension first-assigned-rpc-id {
       description  "YANG statement used to specify the first identifier
                     automatically assigned to a protocol operation
                     within a YANG module. Subsequent protocol
                     operations are assigned sequentially. Values lower
                     than the first-assigned-rpc-id are reserved for
                     manual assignment.";
```

```
      argument "first-rpc-identifier";
    }

    extension first-assigned-notification-id {
      description  "YANG statement used to specify the first identifier
                    automatically assigned to a notification within
                    a Yang module. Subsequent notifications are
                    assigned sequentially. Values lower than the
                    first-assigned-notification-id are reserved for
                    manual assignment.";
      argument "first-notification-identifier";
    }

    extension id {
      description  "YANG statement used to assign a specific identifier
                    to a data node, a protocol operation, a input
                    parameter, a output parameter, a notification
                    or a notification parameter. The argument of this
                    extension contain two information, the path of the
                    specified object and the associated identifier.
                    These two parts are separated by a space.";
      argument "path-vs-id";
    }

    typedef status-code {
      type enumeration {
        enum ok {
          value 0;
          description "The requested edit have been performed
                       successfully.";
        }

        enum error {
          value 1;
          description "Unspecified error.";
        }

        enum malformed {
          value 2;
          description "Malformed CBOR payload.";
        }

        enum invalid {
          value 3;
          description "The value specified in the request can't be
                       apply to the target data node.";
        }
```

```
        enum doesNotExist {
          value 4;
          description "The target data node specified in the request
                       don't exist.";
        }

        enum alreadyExist {
          value 5;
          description "The target data node specified in the request
                       already exist.";
        }

        enum readOnly {
          value 6;
          description "Attempt to update a read-only data node.";
        }
      }
    }

    container error-info {
      description "Optional payload of a client error (CoAP response
                   4.xx) or server error (CoAP response 5.xx).";

      leaf error-code {
        mandatory true;
        type status-code;
      }

      leaf error-text {
        mandatory false;
        type string;
        description "Textual descriptions of the error.";
      }
    }

    list patch-request {
      key "edit-id";

      description
        "This list represents the payload of a patch request message.
         It carry a sequence of edits on a datastore. Each edit MUST
         be applied in ascending order, and all edits MUST be applied.
         If any errors occur, then the target datastore SOULD NOT be
         changed by the patch operation.";

      leaf edit-id {
        mandatory true;
        type uint8;
```

```
        description
          "Error messages returned by the server pertaining to
           a specific edit will be identified by this value.";
      }

      leaf operation {
        mandatory true;
        type enumeration {
          enum "create" {
            value 0;
            description
              "The target data node is created using the supplied
               value, only if it does not already exist.";
          }
          enum "delete" {
            value 1;
            description
              "Delete the target node, only if the data resource
               currently exists, otherwise return an error.";
          }
          enum "merge" {
            value 2;
            description
              "The supplied value is merged with the target
               data node.";
          }
          enum "replace" {
            value 3;
            description
              "The supplied value is used to replace the target
               data node.";
          }
          enum "remove" {
            value 4;
            description
              "Delete the target node if it currently exists.";
          }
          enum insert-first {
            value 5;
            description
              "Insert the supplied value at the beginning of an
               user-ordered-list.";
          }
          enum insert-last {
            value 6;
            description
              "Insert the supplied value at the end of an
               user-ordered-list.";
```

```
          }
          enum insert-before {
            value 7;
            description
              "Insert the supplied value in a user-ordered-list
               just before the entry identified by 'list-entry'
               parameter.";
          }
          enum insert-after {
            value 8;
            description
              "Insert the supplied value in a user-ordered-list
               just after the entry identified by 'list-entry'
               parameter.";
          }
        }
      }

      leaf list-entry {
        when
          "(../operation = 'delete' or ../operation = 'remove' or" +
          " ../operation = 'insert-before' or" +
          " ../operation = 'insert-after' )";
        type binary;
        mandatory true;
        description
          "CBOR array containing an instance identifier used to
           identify an entry within a list.";
      }

      anyxml value {
        when
          "(../operation = 'create' or ../operation = 'merge' or" +
          " ../operation = 'replace' or " +
          " ../operation = 'insert-first' or" +
          " ../operation = 'insert-last' or" +
          " ../operation = 'insert-before' or" +
          " ../operation = 'insert-after')";
        description
          "CBOR map containing the tag and value used for this
           edit operation.";
      }
    }

    list patch-response {
      key "edit-id";
      description
        "This list represents the payload of a patch request response.
```

```
         Each entry contain the status of an operation included in
         the corresponding request message.";

      leaf edit-id {
        mandatory true;
        type uint8;
        description
          "This value is used to match this entry with the
           corresponding entry in the patch-request list";
      }

      leaf edit-status {
        mandatory true;
        type status-code;
      }

      leaf error-text {
        mandatory false;
        type string;
        description "Textual descriptions of the error.";
      }
    }
  }
```

The following identifiers are assigned to the different objects of the ietf-cool module.

DNID	FQDNID	Data node
0	1024	/
1	1025	container /error-info
2	1026	leaf /error-info/error-code
3	1027	leaf /error-info/error-text
4	1028	list /patch-request
5	1029	leaf /patch-request/edit-id
6	1030	leaf /patch-request/operation
7	1031	leaf /patch-request/list-entry
8	1032	anyxml /patch-request/value
9	1033	list /patch-response
10	1034	leaf /patch-response/edit-id
11	1035	leaf /patch-response/edit-status
12	1036	leaf /patch-response/error-text

Appendix B. ietf-cool-library YANG module

This YANG module provides Meta information about modules and sub-modules implemented by the CoOL server.

```
module ietf-cool-library {
  namespace "urn:ietf:ns:cool:library";

  prefix cool;

  description
      "This YANG module provides Meta information about modules and
       sub-modules implemented by the CoOL server";

  revision "2015-09-30" {
    description "Initial revision.";
  }

  typedef module-id {
    description "Registered identifier of a YANG module.";
    type uint32 {
      range "1..1048575";
    }
  }

  list modules {
    key "module-id revision";

    description
      "Each entry represents one module currently
       supported by the server.";

    leaf module-id   {
      mandatory true;
      type module-id;
      description
        "Registered module ID for this YANG module.";
    }

    leaf name {
      type string;
      description
        "Name of this YANG module.";
    }

    leaf revision {
      mandatory true;
      type string {
```

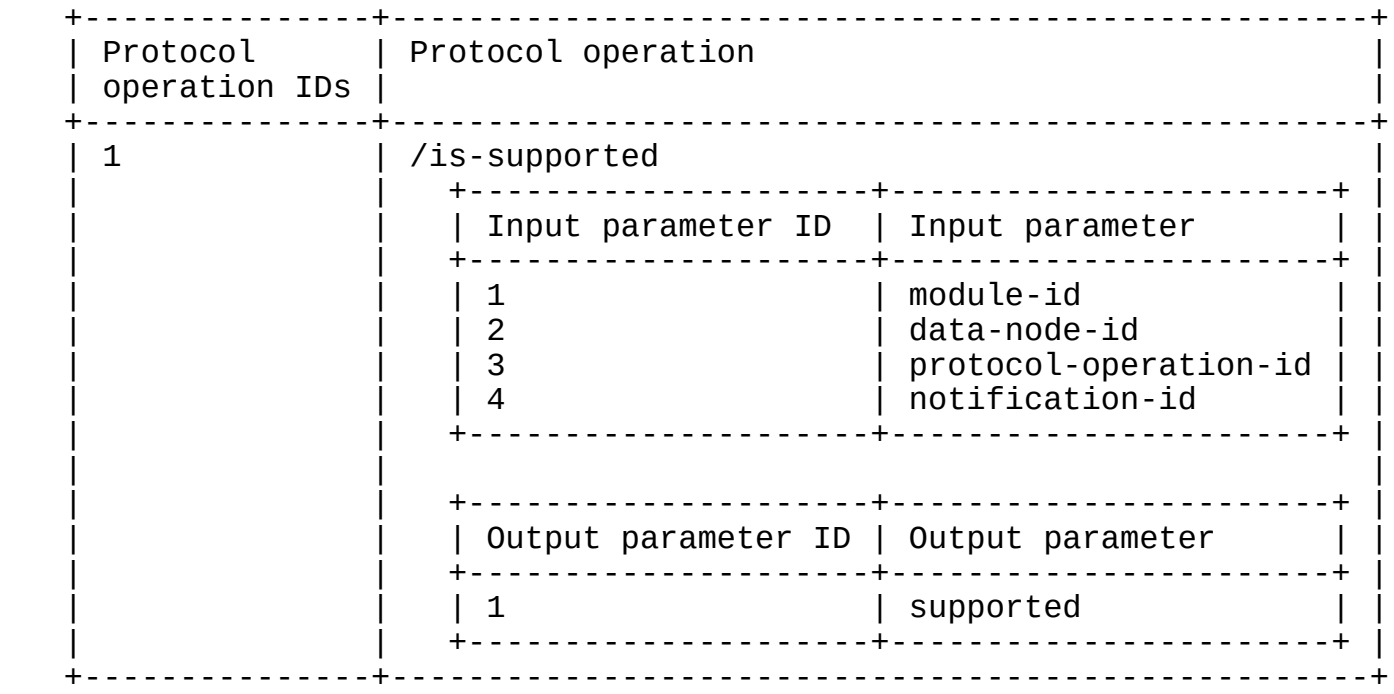
```
        pattern '\d{4}-\d{2}-\d{2}';
      }
      description
        "Revision of this YANG module. An empty string is used if
         no revision statement is present in the YANG module.";
    }

    leaf-list features {
      type string;
      description
        "List of YANG feature names from this module that are
         supported by the server.";
    }

    leaf-list deviations {
      type string;
      description
        "List of YANG deviation module names used by this server to
         modify the conformance of the module associated with this
         entry.";
    }

    list submodules {
      key "name revision";

     description
        "Each entry represents one submodule within the parent
         module.";

      leaf name {
        type string;
        description
          "Name of this YANG submodule.";
      }

      leaf revision {
        mandatory true;
        type string {
          pattern '\d{4}-\d{2}-\d{2}';
        }
        description
          "The YANG submodule revision. An empty string is used if
           no revision statement is present in the YANG submodule.";
      }
    }

    leaf-list data-nodes {
      type uint32;
```

```
        description
          "ID of each data node supported by this module, including
           those defined in sub-modules.";
      }

      leaf-list protocol-operations {
        type uint32;
        description
          "ID of each rpc supported by this module, including those
           defined in sub-modules.";
      }

      leaf-list notifications {
        type uint32;
        description
          "ID of each notification supported by this module,
           including those defined in sub-modules.";
      }
    }

    rpc is-supported {
      description
        "Used to verify if an object is supported by a CoOL server";

      input {

        leaf module-id {
          type module-id;
          mandatory true;
          description "Module ID.";
        }

        choice object {

          case data-node {
            leaf data-node-id {
              type uint16 {
                range "1..1023";
              }
              mandatory true;
              description "Data node ID.";
            }
          }

          case protocol-operation {
            leaf protocol-operation-id {
              type uint16;
              mandatory true;
```

```
            description "Protocol operation ID.";
          }
        }

        case notification {
          leaf notification-id {
            type uint16;
            mandatory true;
            description "Notification ID.";
          }
        }
      }
    }
    output {
      leaf supported {
        type boolean;
        mandatory true;
        description
          "Return true is the specified object is supported.";
      }
    }
  }
}
```

The following identifiers are assigned to the different objects of the cool-library module.

DNID	FQDNID	Data node
0	2048	/
1	2049	list /modules
2	2050	leaf /modules/module-id
3	2051	leaf /modules/name
4	2052	leaf /modules/revision
5	2053	leaf-list /modules/features
6	2054	leaf-list /modules/deviations
7	2055	list /modules/submodules
8	2056	leaf /modules/submodules/name
9	2057	leaf /modules/submodules/revision
10	2058	leaf-list /modules/data-nodes
11	2059	leaf-list /modules/protocol-operations
12	2060	leaf-list /modules/notifications



Authors' Addresses

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

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

Alexander Pelov (editor)
Acklio
2 Rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Phone: +33299127004
Email: a@ackl.io

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

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

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

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