

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641, 7959 (if approved)
Intended status: Standards Track
Expires: May 7, 2020

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB
November 04, 2019

Group Communication for the Constrained Application Protocol (CoAP)
draft-dijk-core-groupcomm-bis-02

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, using UDP/IP multicast as the underlying data transport. Both unsecured and secured CoAP group communication are specified. Security is achieved by use of the Group Object Security for Constrained RESTful Environments (Group OSCORE) protocol. The target application area of this specification is any group communication use cases that involve resource-constrained network nodes. The most common of such use cases are listed in this document.

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 May 7, 2020.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	3
1.1. Scope	4
1.2. Terminology	4
2. General Group Communication Operation	4
2.1. Group Configuration	5
2.1.1. Group Definition	5
2.1.2. Group Naming	5
2.1.3. Group Creation and Membership	6
2.1.4. Group Maintenance	7
2.2. CoAP Usage	7
2.2.1. Request/Response Model	7
2.2.2. Port and URI Path Selection	10
2.2.3. Proxy Operation	10
2.2.4. Congestion Control	12
2.2.5. Observing Resources	13
2.2.6. Block-Wise Transfer	15
2.3. Transport	15
2.3.1. UDP/IPv6 Multicast Transport	15
2.3.2. UDP/IPv4 Multicast Transport	16
2.3.3. 6LoWPAN	16
2.4. Interworking with Other Protocols	16
2.4.1. MLD/MLDv2/IGMP/IGMPv3	16
2.4.2. RPL	17
2.4.3. MPL	18
3. Unsecured Group Communication	18
4. Secured Group Communication using Group OSCORE	18
4.1. Secure Group Maintenance	20
5. Security Considerations	21
5.1. CoAP NoSec Mode	21
5.2. Group OSCORE	21
5.2.1. Group Key Management	22
5.2.2. Source Authentication	22
5.2.3. Counteraction of Attacks	23
5.3. Use of CoAP No Response Option	23
5.4. 6LoWPAN	23
5.5. Wi-Fi	24
5.6. Monitoring	24
5.6.1. General Monitoring	24

5.6.2. Pervasive Monitoring	24
6. IANA Considerations	25
7. References	25
7.1. Normative References	25
7.2. Informative References	27
Appendix A. Use Cases	29
A.1. Discovery	29
A.1.1. Distributed Device Discovery	29
A.1.2. Distributed Service Discovery	30
A.1.3. Directory Discovery	30
A.2. Operational Phase	30
A.2.1. Actuator Group Control	31
A.2.2. Device Group Status Request	31
A.2.3. Network-wide Query	31
A.2.4. Network-wide / Group Notification	31
A.3. Software Update	32
Acknowledgments	32
Authors' Addresses	32

1. Introduction

This document specifies group communication using the Constrained Application Protocol (CoAP) [RFC7252] together with UDP/IP multicast. CoAP is a RESTful communication protocol that is used in resource-constrained nodes, and in resource-constrained networks where packet sizes should be small. This area of use is summarized as Constrained RESTful Environments (CoRE).

One-to-many group communication can be achieved in CoAP, by a client using UDP/IP multicast data transport to send multicast CoAP request messages. In response, each server in the addressed group sends a response message back to the client over UDP/IP unicast. Notable CoAP implementations supporting group communication include the framework "Eclipse Californium" 2.0.x [Californium] from the Eclipse Foundation and the "Implementation of CoAP Server & Client in Go" [Go-OCF] from the Open Connectivity Foundation (OCF).

Both unsecured and secured CoAP group communication over UDP/IP multicast are specified in this document. Security is achieved by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm], which in turn builds on Object Security for Constrained Restful Environments (OSCORE) [RFC8613]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [RFC7049][RFC8152].

All sections in the body of this document are normative, while appendices are informative. For additional background about use

cases for CoAP group communication in resource-constrained devices and networks, see Appendix A.

1.1. Scope

For group communication, only solutions that use CoAP over UDP/multicast are in the scope of this document. There are alternative methods to achieve group communication using CoAP, for example Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. The alternative methods may be usable for the same or similar use cases as are targeted in this document.

All guidelines in [RFC7390] are imported by this document which replaces [RFC7390] in this respect. Furthermore, this document adds: how to provide security by using Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the default group communication security solution for CoAP; an updated request/response matching rule for multicast CoAP which updates [RFC7252]; the multicast use of CoAP Observe which updates [RFC7641]; and an extension of the multicast use of CoAP block-wise transfers, which updates [RFC7959].

Security solutions for group communication and configuration other than Group OSCORE are not in scope. General principles for secure group configuration are in scope. The experimental group configuration protocol in Section 2.6.2 of [RFC7390] is not in the scope of this document; thus, that remains an experimental protocol. Since application protocols defined on top of CoAP often define their own specific method of group configuration, the experimental protocol of [RFC7390] has not been subject to enough experimentation to warrant a change of this status.

1.2. Terminology

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.

This specification requires readers to be familiar with CoAP [RFC7252] terminology.

2. General Group Communication Operation

The general operation of group communication, applicable for both unsecured and secured operation, is specified in this section by going through the stack from top to bottom. First, group

configuration (e.g. group creation and maintenance which are usually done by an application, user or commissioning entity) is considered in Section 2.1. Then the use of CoAP for group communication including support for protocol extensions (block-wise, Observe, PATCH method) follows in Section 2.2. How CoAP group messages are carried over various transport layers is the subject of Section 2.3. Finally, Section 2.4 covers the interworking of CoAP group communication with other protocols that may operate in the same network.

2.1. Group Configuration

2.1.1. Group Definition

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast requests that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups. Group membership(s) of an endpoint may dynamically change over time. A device sending a CoAP request to a group is not necessarily itself a member of this group: it is only a member if it also has a CoAP server endpoint listening to requests for this CoAP group. For secure group communication, a receiver also requires the security context to successfully decrypt and/or verify group messages in order to be a group member.

A CoAP Group URI has the scheme 'coap' and includes in the authority part either an IP multicast address or a group hostname (e.g., Group Fully Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A Group URI also contains an optional UDP port number in the authority part. Group URIs follow the regular CoAP URI syntax (Section 6 of [RFC7252]).

Besides CoAP groups, that have relevance at the level of networked devices, there can also be application groups defined. An application group has relevance at the application level - for example an application group could denote all lights in an office room or all sensors in a hallway. There can be a one-to-one or a one-to-many relation between CoAP groups and application groups.

2.1.2. Group Naming

For clients, it is RECOMMENDED to use by default an IP multicast address literal in a configured Group URI, instead of a hostname. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is used in the Group URI, it can be uniquely mapped to an IP multicast address via DNS resolution - if DNS client functionality is available in the clients

and the DNS service is supported in the network. Some examples of hierarchical group FQDN naming (and scoping) for a building control application are shown in Section 2.2 of [RFC7390].

Application groups can be named in many ways, e.g. numbers, IDs, strings or URIs. An application group identifier, if used, is typically included in the path component or query component of a Group URI. Appendix A of [I-D.ietf-core-resource-directory] shows registration of application groups into a Resource Directory, along with the CoAP group it maps to.

2.1.3. Group Creation and Membership

To create a CoAP group, a configuring entity defines an IP multicast address (or hostname) for the group and optionally a UDP port number in case it differs from the default CoAP port 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP server listening on the group's associated UDP port. These devices are the group members. The configuring entity can be, for example, a local application with preconfiguration, a user, a cloud service, or a local commissioning tool. Also, the devices sending requests to the group in the role of CoAP clients need to be configured with the same information, even though they are not necessarily group members. One way to configure a client is to supply it with a CoAP Group URI, possibly together with the required security material in case communication is secured in the group.

For unsecure group communication, any CoAP endpoint may become a group member at any time: there is no (central) configuring entity that needs to provide the security material for the group. This means that group creation and membership cannot be tightly controlled.

The IETF does not define a mandatory, standardized protocol to accomplish these steps. [RFC7390] defines an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. This protocol remains experimental. For secure group communication, the part of the process that involves secure distribution of group keys MAY use standardized communication with a Group Manager as defined in Section 4.

The configuration of groups and membership may be performed at different moments in the lifecycle of a device; for example in the factory, at a reseller, on-site during first deployment, or on-site during a system reconfiguration operation.

2.1.4. Group Maintenance

Maintenance of a group includes necessary operations to cope with changes in a system, such as: adding group members, removing group members, reconfiguration of UDP port and/or IP multicast address, reconfiguration of the Group URI, splitting of groups, or merging of groups.

For unsecured group communication (see Section 3), addition/removal of group members is simply done by configuring these devices to start/stop listening to the group IP multicast address, and to start/stop the CoAP server listening to the group IP multicast address and port.

For secured group communication (see Section 4), the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] is mandatory to implement. When using Group OSCORE, CoAP endpoints participating to group communication are also members of a corresponding OSCORE group, and thus share a common set of cryptographic material. Additional related maintenance operations are discussed in Section 4.1.

2.2. CoAP Usage

2.2.1. Request/Response Model

A CoAP client is an endpoint able to transmit CoAP requests and receive CoAP responses. Since the underlying UDP supports multiplexing by means of UDP port number, there can be multiple independent CoAP clients operational on a single host. On each UDP port, an independent CoAP client can be hosted. Each independent CoAP client sends requests that use the associated endpoint's UDP port number as the UDP source port of the request.

All CoAP requests that are sent via IP multicast MUST be Non-confirmable (Section 8.1 of [RFC7252]). The Message ID in an IP multicast CoAP message is used for optional message deduplication by both clients and servers, as detailed in Section 4.5 of [RFC7252].

A server sends back a unicast response to the CoAP group communication request - but it MAY suppress this response if selected so by the server application and if permitted by the rules in this document. The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes, depending on the individual server processing results.

Response suppression controlled by a server SHOULD be performed in a consistent way, such that if a first request leads to a response that

is not suppressed, then a second similar request on the same resource that leads to the same response code is also not suppressed.

The CoAP Option for No Server Response [RFC7967] can be used by a client to influence the response suppression on the server side. It is RECOMMENDED for a server to implement this option only on selected resources where it is useful in the application context.

A CoAP client MAY repeat a multicast request using the same Token value and same Message ID value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request. However, in case one or more servers did receive the initial request but the response to that request was lost, this repeat may not help to retrieve the lost response(s) if the server implements the optional Message ID based deduplication.

A CoAP client MAY also repeat a multicast request using the same Token value but a different Message ID, in which case all servers that received the initial request will again process the repeated request. This is useful in case a client needs to collect more, or even all, responses from group members.

The CoAP client can distinguish the origin of multiple server responses by the source IP address of the UDP message containing the CoAP response and/or any other available application-specific source identifiers contained in the CoAP response. While processing a response, the source endpoint of the response is not exactly matched to the destination endpoint of the request, since for a multicast request these will never match. This is specified in Section 8.2 of [RFC7252]. In case a single client has sent multiple group requests and concurrent CoAP transactions are ongoing, the responses received by that client are matched to a request using the Token value. Due to UDP level multiplexing, the UDP destination port of the response MUST match to the client endpoint's UDP port value, i.e. to the UDP source port of the client's request.

For multicast CoAP requests, there are additional constraints on the reuse of Token values at the client, compared to the unicast case. In the unicast case, receiving a response effectively frees up its Token value for reuse, since no more responses to the same request will follow. However, for multicast CoAP, the number of responses is not bound a priori. Therefore, client cannot use the reception of a response as a trigger to "free up" a Token value for reuse. Moreover, reusing a Token value too early could lead to incorrect response/request matching on the client, and would be a protocol error. Therefore, the time between reuse of Token values used in multicast requests MUST be greater than:

`NON_LIFETIME + MAX_LATENCY + MAX_SERVER_RESPONSE_DELAY`

where `NON_LIFETIME` and `MAX_LATENCY` are defined in Section 4.8 of [RFC7252]. This specification defines `MAX_SERVER_RESPONSE_DELAY` as in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a multicast request to. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. However, CoAP does not define a time limit for the server response delay. Using the default CoAP parameters, the Token reuse time **MUST** be greater than 250 seconds plus `MAX_SERVER_RESPONSE_DELAY`. A preferred solution to meet this requirement is to generate a new unique Token for every new multicast request, such that a Token value is never reused. If a client has to reuse Token values for some reason, and also `MAX_SERVER_RESPONSE_DELAY` is unknown, then using `MAX_SERVER_RESPONSE_DELAY = 250` seconds is a reasonable guideline. The time between Token reuses is in that case set to a value greater than 500 seconds.

Another method to more easily meet the above constraint is to instantiate multiple CoAP clients at multiple UDP ports on the same host. The Token values only have to be unique within the context of a single CoAP client, so using multiple clients can make it easier to meet the constraint.

Since a client sending a multicast request with a Token T will accept multiple responses with the same Token T, there is a risk that a same server sends multiple responses with the same Token T back to the client. For example, this server might not implement the optional CoAP message deduplication based on Message ID, or it might be a malicious/compromised server acting out of specification. To mitigate issues with multiple responses from one server bound to a same multicast request, the client has to ensure that, as long as the the CoAP Token used for a multicast request is retained, at most one response to that request per server is accepted, with the exception of Observe notifications [RFC7641] (see Section 2.2.5).

To this end, upon receiving a response corresponding to a multicast request, the client **MUST** perform the following actions. First, the client checks whether it previously received a valid response to this request from the same originating server of the just-received response. If the check yields a positive match and the response is not an Observe notification (i.e., it does not include an Observe option), the client **SHALL** stop processing the response. Upon eventually freeing up the Token value of a multicast request for possible reuse, the client **MUST** also delete the list of responding servers associated to that request.

2.2.2. Port and URI Path Selection

A CoAP server that is a member of a group listens for CoAP messages on the group's IP multicast address, usually on the CoAP default UDP port 5683, or another non-default UDP port if configured. Regardless of the method for selecting the port number, the same port number MUST be used across all CoAP servers that are members of a group and across all CoAP clients performing the group requests to that group. The URI Path used in the request is preferably a path that is known to be supported across all group members. However there are valid use cases where a request is known to be successful for a subset of the group and those group members for which the request is unsuccessful either ignore the multicast request or respond with an error status code.

Using different ports with the same IP multicast address is an efficient way to create multiple CoAP groups in constrained devices, in case the device's stack only supports a limited number of IP multicast group memberships. However, it must be taken into account that this incurs additional processing overhead on each CoAP server participating in at least one of these groups: messages to groups that are not of interest to the node are only discarded at the higher transport (UDP) layer instead of directly at the network (IP) layer.

Port 5684 is reserved for DTLS-secured CoAP and MUST NOT be used for any CoAP group communication.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group.

2.2.3. Proxy Operation

CoAP (Section 5.7.2 of [RFC7252]) allows a client to request a forward-proxy to process its CoAP request. For this purpose, the client specifies either the request group URI as a string in the Proxy-URI option or alternatively it uses the Proxy-Scheme option with the group URI constructed from the usual Uri-* options. This approach works well for unicast requests. However, there are certain issues and limitations of processing the (unicast) responses to a CoAP group communication request made in this manner through a proxy.

A proxy may buffer all the individual (unicast) responses to a CoAP group communication request and then send back only a single (aggregated) response to the client. However, there are some issues with this aggregation approach:

- o Aggregation of (unicast) responses to a CoAP group communication request in a proxy is difficult. This is because the proxy does not know how many members there are in the group or how many group members will actually respond. Also, the proxy does not know how long to wait before deciding to send back the aggregated response to the client.
- o There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be defined based on the multipart content-format [I-D.ietf-core-multipart-ct] but is not standardized yet currently.

Alternatively, if a proxy does not aggregate responses and follows the CoAP Proxy specification (Section 5.7.2 of [RFC7252]), the proxy would forward all the individual (unicast) responses to a CoAP group communication request to the client. There are also issues with this approach:

- o The client may be confused as it may not have known that the Proxy-URI contained a group URI target. That is, the client that sent a unicast CoAP request to the proxy may be expecting only one (unicast) response. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the application.
- o Each individual CoAP response will appear to originate (based on its IP source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response, unless the server identity is contained as a part of the response payload.

Due to the above issues, a CoAP Proxy SHOULD NOT support processing an IP multicast CoAP request but rather return a 501 (Not Implemented) response in such case. The exception case here (i.e., to support it) is when all the following conditions are met:

- o The CoAP Proxy MUST be explicitly configured (whitelist) to allow proxied IP multicast requests by specific client(s).
- o The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). If a (future) standardized aggregation format is being used, then aggregated responses may be sent.
- o It MUST be known to the person/entity doing the configuration of the proxy, or otherwise verified in some way, that the client

configured in the whitelist supports receiving multiple responses to a proxied unicast CoAP request.

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Section 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type can be used to let the proxy return multiple CoAP responses - each translated to a HTTP response - back to the HTTP client. Of course, in this case the HTTP client needs to be aware that it is receiving this format and needs to be able to decode from it the responses of multiple servers. The above restrictions listed for CoAP Proxies still apply to HTTP-to-CoAP proxies: specifically, the IP address of the sender of each CoAP response cannot be determined from the application/http response.

2.2.4. Congestion Control

CoAP group communication requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore, both the sending of IP multicast requests and the sending of the unicast CoAP responses to these multicast requests should be conservatively controlled.

CoAP [RFC7252] reduces IP multicast-specific congestion risks through the following measures:

- o A server may choose not to respond to an IP multicast request if there is nothing useful to respond to, e.g., error or empty response (see Section 8.2 of [RFC7252]).
- o A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).
- o An IP multicast request MUST be Non-confirmable (Section 8.1 of [RFC7252]).
- o A response to an IP multicast request SHOULD be Non-confirmable (Section 5.2.3 of [RFC7252]).
- o A server does not respond immediately to an IP multicast request and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 of [RFC7252]).

Additional guidelines to reduce congestion risks defined in this document are as follows:

- o A server in an LLN should only support group communication GET for resources that are small. This can consist, for example, in having the payload of the response as limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.
- o A server SHOULD minimize the payload length of a response to a multicast GET on `"/.well-known/core"` by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].
- o A server MAY minimize the payload length of a response to a multicast GET (e.g., on `"/.well-known/core"`) by using CoAP block-wise transfers [RFC7959] in case the payload is long, returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending an IP multicast CoAP request to `"/.well-known/core"` SHOULD support block-wise transfers.
- o A client SHOULD use CoAP group communication with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope if this fulfills the application needs.

2.2.5. Observing Resources

The CoAP Observe Option [RFC7641] is a protocol extension of CoAP, that allows a CoAP client to retrieve a representation of a resource and automatically keep this representation up-to-date over a longer period of time. The client gets notified when the representation has changed. [RFC7641] does not mention whether the Observe Option can be combined with CoAP multicast. This section updates [RFC7641] with the use of the Observe Option in a CoAP multicast GET request and defines normative behavior for both client and server.

Multicast Observe is a useful way to start observing a particular resource on all members of a (multicast) group at the same time. Group members that do not have this particular resource or do not allow the GET method on it will either respond with an error status - 4.04 Not Found or 4.05 Method Not Allowed, respectively - or will silently suppress the response following the rules of Section 2.2.1, depending on server-specific configuration.

A client that sends a multicast GET request with the Observe Option MAY repeat this request using the same Token value and the same

Observe Option value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request to avoid that group members that had already received the initial request would respond again. Note that using the same Message ID in a repeated request will not be helpful in case of loss of a response message, since the server that responded already will consider the repeated request as a duplicate message. On the other hand, if the client uses a different, fresh Message ID in the repeated request, then all the group members that receive this new message will typically respond again, which increases the network load.

A client that sent a multicast GET request with the Observe Option MAY follow up by sending a new unicast CON request with the same Token value and same Observe Option value to a particular server, in order to ensure that the particular server receives the request. This is useful in case a specific group member, that was expected to respond to the initial group request, did not respond to the initial request. The client in this case always uses a Message ID that differs from the initial multicast message.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that a client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]).

Before repeating a request as specified above, the client SHOULD wait for at least the expected round-trip time plus the Leisure time period defined in Section 8.2 of [RFC7252], to give the server time to respond.

A server that receives a legitimate GET request with the Observe Option, for which request processing is successful, SHOULD NOT suppress the response to this request, because the client is obviously interested in the resource representation. A server that adds a client to the list of observers for a resource due to an Observe request MUST NOT suppress the response to this request.

A server SHOULD have a mechanism to verify liveness of its observing clients and the continued interest of these clients in receiving the observe notifications. This can be implemented by sending notifications occasionally using a Confirmable message. See Section 4.5 of [RFC7641] for details. This requirement overrides the regular behavior of sending Non-Confirmable notifications in response to a Non-Confirmable request.

For observing a group of servers through a CoAP-to-CoAP proxy or HTTP-CoAP proxy, the limitations stated in Section 2.2.3 apply.

2.2.6. Block-Wise Transfer

Section 2.8 of [RFC7959] specifies how a client can use block-wise transfer (Block2 Option) in a multicast GET request to limit the size of the initial response of each server. The client has to use unicast for any further requests, separately addressing each different server, in order to retrieve more blocks of the resource from that server, if any. Also, a server (group member) that needs to respond to a multicast request with a particularly large resource can use block-wise transfer (Block2 Option) at its own initiative, to limit the size of the initial response. Again, a client would have to use unicast for any further requests to retrieve more blocks of the resource.

A solution for multicast block-wise transfer using the Block1 Option is not specified in [RFC7959] nor in the present document. Such a solution would be useful for multicast PUT/POST/PATCH/iPATCH requests, to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. Multicast usage of Block1 is non-trivial due to potential message loss (leading to missing blocks or missing confirmations), and potential diverging block size preferences of different members of the multicast group.

2.3. Transport

In this document only UDP is considered as a transport protocol, both over IPv4 and IPv6. Therefore, [RFC8323] (CoAP over TCP, TLS, and WebSockets) is not in scope as a transport for group communication.

2.3.1. UDP/IPv6 Multicast Transport

CoAP group communication can use UDP over IPv6 as a transport protocol, provided that IPv6 multicast is enabled. IPv6 multicast MAY be supported in a network only for a limited scope. For example, Section 2.4.2 describes the potential limited support of RPL for multicast, depending on how the protocol is configured.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported as per Section 7.1 and 12.8 of [RFC7252] for the "All CoAP Nodes" multicast group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" groups with at least link-local (2), admin-local (4) and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 2.3.3) SHOULD also support the realm-local (3) scope.

Note that a client sending an IPv4 multicast CoAP message to a port that is not supported by the server will not receive an ICMP Port Unreachable error message from that server, because the server does not send it in this case, per Section 3.2.2 of [RFC1122].

2.3.2. UDP/IPv4 Multicast Transport

CoAP group communication can use UDP over IPv4 as a transport protocol, provided that IPv4 multicast is enabled. For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported as per Section 7.1 and 12.8 of [RFC7252], for the "All CoAP Nodes" IPv4 multicast group.

Note that a client sending an IPv6 multicast CoAP message to a port that is not supported by the server will not receive an ICMPv6 Port Unreachable error message from that server, because the server does not send it in this case, per Section 2.4 of [RFC4443].

2.3.3. 6LoWPAN

In 6LoWPAN [RFC4944] networks, IPv6 packets (up to 1280 bytes) may be fragmented into smaller IEEE 802.15.4 MAC frames (up to 127 bytes), if the packet size requires this. Every 6LoWPAN IPv6 router that receives a multi-fragment packet reassembles the packet and refragments it upon transmission. Since the loss of a single fragment implies the loss of the entire IPv6 packet, the performance in terms of packet loss and throughput of multi-fragment multicast IPv6 packets is typically far worse than the performance of single-fragment IPv6 multicast packets. For this reason, a CoAP request sent over multicast in 6LoWPAN networks SHOULD be sized in such a way that it fits in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast groups can be defined with realm-local scope [RFC7346]. Such a realm-local group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that group does not propagate beyond the 6LoWPAN network segment where the request originated. For example, a multicast discovery request can be sent to the realm-local "All CoAP Nodes" IPv6 multicast group (see Section 2.3.1) in order to discover only CoAP servers on the local 6LoWPAN network.

2.4. Interworking with Other Protocols

2.4.1. MLD/MLDv2/IGMP/IGMPv3

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP multicast routing/forwarding protocol being used in their network. When such a host needs to join a

specific (CoAP) multicast group, it requires a way to signal to IP multicast routers which IP multicast address(es) it needs to listen to.

The MLDv2 protocol [RFC3810] is the standard IPv6 method to achieve this; therefore, this method SHOULD be used by group members to subscribe to the multicast group IPv6 address, on IPv6 networks that support it. CoAP server nodes then act in the role of MLD Multicast Address Listener. Constrained IPv6 networks that implement either RPL (see Section 2.4.2) or MPL (see Section 2.4.3) typically do not support MLD as they have their own mechanisms defined.

The IGMPv3 protocol [RFC3376] is the standard IPv4 method to signal multicast group subscriptions. This SHOULD be used by group members to subscribe to their multicast group IPv4 address on IPv4 networks.

The guidelines from [RFC6636] on the tuning of MLD for mobile and wireless networks may be useful when implementing MLD in LLNs.

2.4.2. RPL

RPL [RFC6550] is an IPv6 based routing protocol suitable for low-power, lossy networks (LLNs). In such a context, CoAP is often used as an application protocol.

If RPL is used in a network for routing and its optional multicast support is disabled, there will be no IP multicast routing available. Any IPv6 multicast packets in this case will not propagate beyond a single hop (to direct neighbors in the LLN). This implies that any CoAP group communication request will be delivered to link-local nodes only, for any scope value ≥ 2 used in the IPv6 destination address.

RPL supports (see Section 12 of [RFC6550]) advertisement of IP multicast destinations using Destination Advertisement Object (DAO) messages and subsequent routing of multicast IPv6 packets based on this. It requires the RPL mode of operation to be 3 (Storing mode with multicast support).

In this mode, RPL DAO can be used by a CoAP node that is either an RPL router or RPL Leaf Node, to advertise its IP multicast group membership to parent RPL routers. Then, RPL will route any IP multicast CoAP requests over multiple hops to those CoAP servers that are group members.

The same DAO mechanism can be used to convey IP multicast group membership information to an edge router (e.g., 6LBR), in case the edge router is also the root of the RPL Destination-Oriented Directed

Acyclic Graph (DODAG). This is useful because the edge router then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet. In LLNs, such ingress filtering helps to avoid congestion of the resource-constrained network segment, due to IP multicast traffic from the high-speed backbone IP network.

2.4.3. MPL

The Multicast Protocol for Low-Power and Lossy Networks (MPL) [RFC7731] can be used for propagation of IPv6 multicast packets throughout a defined network domain, over multiple hops. MPL is designed to work in LLNs. MPL defines the protocol for a predefined group of MPL Forwarders to collectively distribute IPv6 multicast packets throughout their MPL Domain. An MPL Forwarder may be associated to multiple MPL Domains at the same time. Non-Forwarders will receive IPv6 multicast packets from one or more of their neighboring Forwarders. Therefore, MPL can be used to propagate a CoAP multicast request to all group members.

However, a CoAP multicast request to a group that originated outside of the MPL Domain will not be propagated by MPL - unless an MPL Forwarder is explicitly configured as an ingress point that introduces external multicast packets into the MPL Domain. Such an ingress point could be located on an edge router (e.g., 6LBR). The method to configure which multicast groups are to be propagated into the MPL Domain could be:

- o Manual configuration on the ingress MPL Forwarder.
- o A protocol to register multicast groups at an ingress MPL Forwarder. This could be a protocol offering features similar to MLDv2.

3. Unsecured Group Communication

CoAP group communication can operate in CoAP NoSec (No Security) mode, without using application-layer and transport-layer security mechanisms. The NoSec mode uses the "coap" scheme, and is defined in Section 9 of [RFC7252]. Before using this mode of operation, the security implications (Section 5.1) must be well understood.

4. Secured Group Communication using Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end encryption, integrity and replay protection of CoAP messages exchanged between two CoAP endpoints. These can act both as CoAP

Client as well as CoAP Server, and share an OSCORE Security Context used to protect and verify exchanged messages. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP.

OSCORE uses COSE [RFC8152] to perform encryption, signing and Message Authentication Code operations, and to efficiently encode the result as a COSE object. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message, by using an Authenticated Encryption with Associated Data (AEAD) algorithm.

OSCORE makes it possible to selectively protect different parts of a CoAP message in different ways, so still allowing intermediaries (e.g., CoAP proxies) to perform their intended functionalities. That is, some message parts are encrypted and integrity protected; other parts are only integrity protected to be accessible to, but not modifiable by, proxies; and some parts are kept as plain content to be both accessible to and modifiable by proxies. Such differences especially concern the CoAP options included in the unprotected message.

Group OSCORE [I-D.ietf-core-oscore-groupcomm] builds on OSCORE, and provides end-to-end security of CoAP messages exchanged between members of an OSCORE group, while fulfilling the same security requirements.

In particular, Group OSCORE protects CoAP requests sent over IP multicast by a CoAP client, as well as multiple corresponding CoAP responses sent over IP unicast by different CoAP servers. However, the same keying material can also be used to protect CoAP requests sent over IP unicast to a single CoAP server in the OSCORE group, as well as the corresponding responses.

Group OSCORE uses digital signatures to ensure source authentication of all messages exchanged within the OSCORE group. That is, sender devices sign their outgoing messages by means of their own private key, and embed the signature in the protected CoAP message.

A Group Manager is responsible for one or multiple OSCORE groups. In particular, the Group Manager acts as repository of public keys of group members; manages, renews and provides keying material in the group; and handles the join process of new group members.

As recommended in [I-D.ietf-core-oscore-groupcomm], a CoAP endpoint can join an OSCORE group by using the method described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework

for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

A CoAP endpoint can discover OSCORE groups and retrieve information to join them through their Group Managers by using the method described in [I-D.tiloca-core-oscore-discovery] and based on the CoRE Resource Directory [I-D.ietf-core-resource-directory].

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1.1 and using secure group communication is associated to an OSCORE group, which includes:

- o All members of the CoAP group, i.e. the CoAP endpoints configured (also) as CoAP servers and listening to the group's multicast IP address.
- o All further CoAP endpoints configured only as CoAP clients, that send (multicast) CoAP requests to the CoAP group.

4.1. Secure Group Maintenance

Additional key management operations on the OSCORE group are required, depending also on the security requirements of the application (see Section 5.2). That is:

- o Adding new members to a CoAP group or enabling new client-only endpoints to interact with that group require also that each of such members/endpoints join the corresponding OSCORE group. By doing so, they are securely provided with the necessary cryptographic material. In case backward security is needed, this also requires to first renew such material and distribute it to the current members/endpoints, before new ones are added and join the OSCORE group.
- o In case forward security is needed, removing members from a CoAP group or stopping client-only endpoints from interacting with that group requires removing such members/endpoints from the corresponding OSCORE group. To this end, new cryptographic material is generated and securely distributed only to the remaining members/endpoints. This ensures that only the members/endpoints intended to remain are able to continue participating to secure group communication, while the evicted ones are not able to.

The key management operations mentioned above are entrusted to the Group Manager responsible for the OSCORE group [I-D.ietf-core-oscore-groupcomm], and it is RECOMMENDED to perform

them according to the approach described in [I-D.ietf-ace-key-groupcomm-oscore].

5. Security Considerations

This section provides security considerations for CoAP group communication using IP multicast.

5.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast.

Thus, for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems), it is NOT RECOMMENDED to deploy CoAP group communication in NoSec mode.

Without application-layer security, CoAP group communication SHOULD only be deployed in applications that are non-critical, and that do not involve or may have an impact on sensitive data and personal sphere. These include, e.g., read-only temperature sensors deployed in non-sensitive environments, where the client reads out the values but does not use the data to control actuators or to base an important decision on.

Discovery of devices and resources is a typical use case where NoSec mode is applied, since the devices involved do not have yet configured any mutual security relations at the time the discovery takes place.

5.2. Group OSCORE

Group OSCORE provides end-to-end application-level security. This has many desirable properties, including maintaining security assurances while forwarding traffic through intermediaries (proxies). Application-level security also tends to more cleanly separate security from the dynamics of group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

For sensitive and mission-critical applications, CoAP group communication MUST be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 8 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

5.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group keying material, namely group rekeying, should be adopted in OSCORE groups. In particular, the key management scheme should preserve backward and forward security in the OSCORE group, if the application requires so (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]).

Group policies should also take into account the time that the key management scheme requires to rekey the group, on one hand, and the expected frequency of group membership changes, i.e. nodes' joining and leaving, on the other hand.

In fact, it may be desirable to not rekey the group upon every single membership change, in case members' joining and leaving are frequent, and at the same time a single group rekeying instance takes a non negligible time to complete.

In such a case, the Group Manager may consider to rekey the group, e.g., after a minimum number of nodes has joined or left the group within a pre-defined time interval, or according to communication patterns with predictable intervals of network inactivity. This would prevent paralyzing communications in the group, when a slow rekeying scheme is used and frequently invoked.

This comes at the cost of not continuously preserving backward and forward security, since group rekeying might not occur upon every single group membership change. That is, latest joined nodes would have access to the key material used prior to their join, and thus be able to access past group communications protected with that key material. Similarly, until the group is rekeyed, latest left nodes would preserve access to group communications protected with the retained key material.

5.2.2. Source Authentication

CoAP endpoints using Group OSCORE countersign their outgoing messages, by means of the countersignature algorithm used in the OSCORE group. This ensures source authentication of messages exchanged by CoAP endpoints through CoAP group communication. In fact, it allows to verify that a received message has actually been originated by a specific and identified member of the OSCORE group.

Appendix F of [I-D.ietf-core-oscore-groupcomm] discusses a number of cases where a recipient CoAP endpoint may skip the verification of countersignatures, possibly on a per-message basis. However, this is NOT RECOMMENDED. That is, a CoAP endpoint receiving a message secured with Group OSCORE SHOULD always verify the countersignature.

5.2.3. Counteraction of Attacks

Group OSCORE addresses security attacks mentioned in Sections 11.2–11.6 of [RFC7252], with particular reference to their execution over IP multicast. That is: it provides confidentiality and integrity of request/response data through proxies also in multicast settings; it prevents amplification attacks carried out through responses to injected requests over IP multicast; it limits the impact of attacks based on IP spoofing; it prevents cross-protocol attacks; it derives the group key material from, among other things, a Master Secret securely generated by the Group Manager and provided to CoAP endpoints upon their joining of the OSCORE group; countersignatures assure source authentication of exchanged CoAP messages, and hence prevent a group member to be used for subverting security in the whole group.

5.3. Use of CoAP No Response Option

The CoAP No Server Response Option [RFC7967] could be misused by a malicious client to evoke as much responses from servers to a multicast request as possible, by using the value '0' - Interested in all responses. This can even override the default behaviour of a CoAP server to suppress the response in case there is nothing of interest to respond with. Therefore, this option can be used to perform an amplification attack. A proposed mitigation is to only allow this Option to relax the standard suppression rules for a resource in case the Option is sent by an authenticated client. If sent by an unauthenticated client, the Option can be used to expand the classes of responses suppressed compared to the default rules but not to reduce the classes of responses suppressed.

5.4. 6LoWPAN

In a 6LoWPAN network, a multicast IPv6 packet may be fragmented prior to transmission. A 6LoWPAN Router that forwards a fragmented packet can have a relatively high impact on the occupation of the wireless channel and on the memory load of the local node due to packet buffer occupation. For example, the MPL [RFC7731] protocol requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. If only one of the fragments is not received correctly by an MPL Forwarder, the receiver needs to discard all received fragments and it needs to receive all the packet fragments again on a future occasion.

For these reasons, a fragmented IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack. See Section 11.3 of [RFC7252] for more details on amplification. To mitigate the risk, applications sending multicast IPv6 requests to

6LoWPAN hosted CoAP servers SHOULD limit the size of the request to avoid 6LoWPAN fragmentation. A 6LoWPAN Router or multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets. Also, a 6LoWPAN Border Router SHOULD implement multicast packet filtering to prevent unwanted multicast traffic from entering a 6LoWPAN network from the outside. For example, it could filter out all multicast packet for which there is no known multicast listener on the 6LoWPAN network.

5.5. Wi-Fi

In a home automation scenario using Wi-Fi, Wi-Fi security should be enabled to prevent rogue nodes from joining. The Customer Premises Equipment (CPE) that enables access to the Internet should also have its IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of IP multicast transmissions and listeners should be site-local (5) or smaller. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

5.6. Monitoring

5.6.1. General Monitoring

CoAP group communication can be used to control a set of related devices: for example, simultaneously turn on all the lights in a room. This intrinsically exposes the group to some unique monitoring risks that devices not in a group are not as vulnerable to. For example, assume an attacker is able to physically see a set of lights turn on in a room. Then the attacker can correlate an observed CoAP group communication message to the observed coordinated group action – even if the CoAP message is (partly) encrypted. This will give the attacker side-channel information to plan further attacks (e.g., by determining the members of the group some network topology information may be deduced).

5.6.2. Pervasive Monitoring

A key additional threat consideration for group communication is pervasive monitoring [RFC7258]. CoAP group communication solutions that are built on top of IP multicast need to pay particular heed to these dangers. This is because IP multicast is easier to intercept (and to secretly record) compared to IP unicast. Also, CoAP traffic is meant for the Internet of Things. This means that CoAP multicast may be used for the control and monitoring of critical infrastructure (e.g., lights, alarms, etc.) that may be prime targets for attack.

For example, an attacker may attempt to record all the CoAP traffic going over a smart grid (i.e., networked electrical utility) and try to determine critical nodes for further attacks. For example, the source node (controller) sends out CoAP group communication messages which easily identifies it as a controller. CoAP multicast traffic is inherently more vulnerable (compared to unicast) as the same packet may be replicated over many links, leading to a higher probability of packet capture by a pervasive monitoring system.

One mitigation is to restrict the scope of IP multicast to the minimal scope that fulfills the application need. Thus, for example, site-local IP multicast scope is always preferred over global scope IP multicast if this fulfills the application needs.

Even if all CoAP multicast traffic is encrypted/protected, an attacker may still attempt to capture this traffic and perform an off-line attack in the future.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP",
draft-ietf-core-oscore-groupcomm-05 (work in progress),
July 2019.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -
Communication Layers", STD 3, RFC 1122,
DOI 10.17487/RFC1122, October 1989,
<<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A.
Thyagarajan, "Internet Group Management Protocol, Version
3", RFC 3376, DOI 10.17487/RFC3376, October 2002,
<<https://www.rfc-editor.org/info/rfc3376>>.

- [RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <<https://www.rfc-editor.org/info/rfc3810>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/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/info/rfc7641>>.
- [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/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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/info/rfc8613>>.

7.2. Informative References

- [Californium]
Eclipse Foundation, "Eclipse Californium", March 2019, <<https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>>.
- [Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <<https://github.com/go-ocf/go-coap>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-02 (work in progress), July 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-09 (work in progress), September 2019.
- [I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-04 (work in progress), August 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.

- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-03 (work in progress), July 2019.
- [RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <<https://www.rfc-editor.org/info/rfc6092>>.
- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, DOI 10.17487/RFC6636, May 2012, <<https://www.rfc-editor.org/info/rfc6636>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <<https://www.rfc-editor.org/info/rfc7346>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7731] Hui, J. and R. Kelsey, "Multicast Protocol for Low-Power and Lossy Networks (MPL)", RFC 7731, DOI 10.17487/RFC7731, February 2016, <<https://www.rfc-editor.org/info/rfc7731>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.

Appendix A. Use Cases

To illustrate where and how CoAP-based group communication can be used, this section summarizes the most common use cases. These use cases include both secured and non-secured CoAP usage. Each subsection below covers one particular category of use cases for CoRE. Within each category, a use case may cover multiple application areas such as home IoT, commercial building IoT (sensing and control), industrial IoT/control, or environmental sensing.

A.1. Discovery

Discovery of physical devices in a network, or discovery of information entities hosted on network devices, are operations that are usually required in a system during the phases of setup or (re)configuration. When a discovery use case involves devices that need to interact without having been configured previously with a common security context, unsecured CoAP communication is typically used. Discovery may involve a request to a directory server, which provides services to aid clients in the discovery process. One particular type of directory server is the CoRE Resource Directory [I-D.ietf-core-resource-directory]; and there may be other types of directories that can be used with CoAP.

A.1.1. Distributed Device Discovery

Device discovery is the discovery and identification of networked devices - optionally only devices of a particular class, type, model, or brand. Group communication is used for distributed device discovery, if a central directory server is not used. Typically in distributed device discovery, a multicast request is sent to a particular address (or address range) and multicast scope of interest, and any devices configured to be discoverable will respond back. For the alternative solution of centralized device discovery a central directory server is accessed through unicast, in which case group communication is not needed. This requires that the address of the central directory is either preconfigured in each device or configured during operation using a protocol.

In CoAP, device discovery can be implemented by CoAP resource discovery requesting (GET) a particular resource that the sought device class, type, model or brand is known to respond to. It can

also be implemented using CoAP resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690] to find these particular resources. Also, a multicast GET request to /.well-known/core can be used to discover all CoAP devices.

A.1.2. Distributed Service Discovery

Service discovery is the discovery and identification of particular services hosted on network devices. Services can be identified by one or more parameters such as ID, name, protocol, version and/or type. Distributed service discovery involves group communication to reach individual devices hosting a particular service; with a central directory server not being used.

In CoAP, services are represented as resources and service discovery is implemented using resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690].

A.1.3. Directory Discovery

This use case is a specific sub-case of Distributed Service Discovery (Appendix A.1.2), in which a device needs to identify the location of a Directory on the network to which it can e.g. register its own offered services, or to which it can perform queries to identify and locate other devices/services it needs to access on the network. Section 3.3 of [RFC7390] shows an example of discovering a CoRE Resource Directory using CoAP group communication. As defined in [I-D.ietf-core-resource-directory], a resource directory is a web entity that stores information about web resources and implements REST interfaces for registration and lookup of those resources. For example, a device can register itself to a resource directory to let it be found by other devices and/or applications.

A.2. Operational Phase

Operational phase use cases describe those operations that occur most frequently in a networked system, during its operational lifetime and regular operation. Regular usage is when the applications on networked devices perform the tasks they were designed for and exchange of application-related data using group communication occurs. Processes like system reconfiguration, group changes, system/device setup, extra group security changes, etc. are not part of regular operation.

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and 3.5 of [RFC7390] show examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g. as "all temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

In some cases a whole network or subnet of multiple IP devices needs to be queried for status or other information. This is similar to the previous use case except that the device group is not defined in terms of its function/type but in terms of its network location. Technically this is also similar to distributed service discovery (Appendix A.1.2) where a query is processed by all devices on a network - except that the query is not about services offered by the device, but rather specific operational data is requested.

A.2.4. Network-wide / Group Notification

In some cases a whole network, or subnet of multiple IP devices, or a specific target group needs to be notified of a status change or other information. This is similar to the previous two use cases except that the recipients are not expected to respond with some information. Unreliable notification can be acceptable in some use cases, in which a recipient does not respond with a confirmation of having received the notification. In such a case, the receiving CoAP server does not have to create a CoAP response. If the sender needs confirmation of reception, the CoAP servers can be configured for that resource to respond with a 2.xx success status after processing a notification request successfully.

A.3. Software Update

Multicast can be useful to efficiently distribute new software (firmware, image, application, etc.) to a group of multiple devices. In this case, the group is defined in terms of device type: all devices in the target group are known to be capable of installing and running the new software. The software is distributed as a series of smaller blocks that are collected by all devices and stored in memory. All devices in the target group are usually responsible for integrity verification of the received software; which can be done per-block or for the entire software image once all blocks have been received. Due to the inherent unreliability of CoAP multicast, there needs to be a backup mechanism (e.g. implemented using CoAP unicast) by which a device can individually request missing blocks of a whole software image/entity. Prior to multicast software update, the group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Acknowledgments

The authors sincerely thank Thomas Fossati and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Esko Dijk
IoTconsultancy.nl

Utrecht
The Netherlands

Email: esko.dijk@iotconsultancy.nl

Chonggang Wang
InterDigital
1001 E Hector St, Suite 300
Conshohocken PA 19428
United States

Email: Chonggang.Wang@InterDigital.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 2, 2020

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
September 30, 2019

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-09

Abstract

The Constrained Application Protocol (CoAP), and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines a publish-subscribe Broker for CoAP that extends the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

There is work in progress to resolve some of the transfer layer issues by using a more RESTful approach.

Please see <https://github.com/core-wg/pubsub/blob/master/proposal.txt>

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 April 2, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Architecture	4
3.1. CoAP Pub/sub Architecture	4
3.2. CoAP Pub/sub Broker	5
3.3. CoAP Pub/sub Client	5
3.4. CoAP Pub/sub Topic	5
3.5. Brokerless Pub/sub	6
4. CoAP Pub/sub REST API	7
4.1. DISCOVERY	7
4.2. CREATE	9
4.3. PUBLISH	11
4.4. SUBSCRIBE	14
4.5. UNSUBSCRIBE	16
4.6. READ	17
4.7. REMOVE	19
5. CoAP Pub/sub Operation with Resource Directory	20
6. Sleep-Wake Operation	21
7. Simple Flow Control	21
8. Security Considerations	21
9. IANA Considerations	23
9.1. Resource Type value 'core.ps'	23
9.2. Resource Type value 'core.ps.discover'	23
10. Acknowledgements	23
11. References	23
11.1. Normative References	23
11.2. Informative References	24
Authors' Addresses	25

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources.

Depending on the situation the same device may act either as a server, a client, or both.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited reachability because they spend most of their time in a sleeping state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

For some applications the client/server and request/response communication model is not optimal but publish-subscribe communication with potentially many senders and/or receivers and communication via topics rather than directly with endpoints may fit better.

This document specifies simple extensions to CoAP for enabling publish-subscribe communication using a Broker node that enables store-and-forward messaging between two or more nodes. This model facilitates communication of nodes with limited reachability, enables simple many-to-many communication, and eases integration with other publish-subscribe systems.

2. Terminology

`{::boilerplate bcp14}`

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST API defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a Broker and potential receivers can subscribe to the Broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a Broker and publications are delivered by the Broker to subscribed receivers.

CoAP pub/sub service: A group of REST resources, as defined in this document, which together implement the required features of this specification.

CoAP pub/sub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The Broker can also temporarily store publications to satisfy future subscriptions and pending notifications.

CoAP pub/sub Client: A CoAP client which is capable of publish or subscribe operations as defined in this specification.

Topic: A unique identifier for a particular item being published and/or subscribed to. A Broker uses the topics to match subscriptions to publications. A reference to a Topic on a Broker is a valid CoAP URI as defined in [RFC7252]

3. Architecture

3.1. CoAP Pub/sub Architecture

Figure 1 shows the architecture of a CoAP pub/sub service. CoAP pub/sub Clients interact with a CoAP pub/sub Broker through the CoAP pub/sub REST API which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pub/sub Broker performs a store-and-forward of state update representations between certain CoAP pub/sub Clients. Clients Subscribe to topics upon which representations are Published by other Clients, which are forwarded by the Broker to the subscribing clients. A CoAP pub/sub Broker may be used as a REST resource proxy, retaining the last published representation to supply in response to Read requests from Clients.

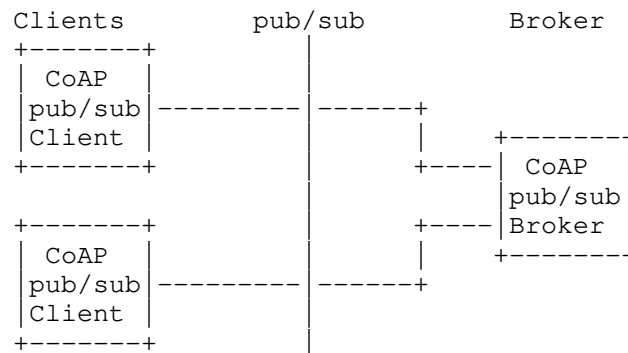


Figure 1: CoAP pub/sub Architecture

3.2. CoAP Pub/sub Broker

A CoAP pub/sub Broker is a CoAP Server that exposes a REST API for clients to use to initiate publish-subscribe interactions. Avoiding the need for direct reachability between clients, the Broker only needs to be reachable from all clients. The Broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resource services, and potentially buffer messages, on behalf of the clients.

3.3. CoAP Pub/sub Client

A CoAP pub/sub Client interacts with a CoAP pub/sub Broker using the CoAP pub/sub REST API defined in this document. Clients initiate interactions with a CoAP pub/sub Broker. A data source (e.g., sensor clients) can publish state updates to the Broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the Broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and data sinks.

3.4. CoAP Pub/sub Topic

The clients and Broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. `"/sensors/weather/barometer/pressure"` or `"/EP-33543/sen/3303/0/5700"`. The topics are hosted by a Broker and all the clients using the Broker share the same namespace for topics.

Every CoAP pub/sub topic has an associated link, consisting of a reference path on the Broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero

or more stored representations and a content-format specified in the link. A CoAP pub/sub topic value may alternatively consist of a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format. Sub-topics are also topics and may have their own sub-topics, forming a tree structure of unique paths that is implemented using URIs. The full URI of a topic includes a scheme and authority for the Broker, for example "coaps://192.0.2.13:5684/EP-33543/sen/3303/0/5700".

A Topic may have a lifetime defined by using the CoAP Max-Age option on topic creation, or on publish operations to the topic. The lifetime is refreshed each time a representation is published to the topic. If the lifetime expires, the topic is removed from discovery responses, returns errors on subscription, and any outstanding subscriptions are cancelled.

3.5. Brokerless Pub/sub

In some use cases, it is desireable to use pub/sub semantics for peer-to-peer communication, but it is not feasible or desireable to include a separate node on the network to serve as a Broker. In other use cases, it is desireable to enable one-way-only communication, such as sensors pushing updates to a service.

Figure 2 shows an arrangement for using CoAP pub/sub in a "Brokerless" configuration between peer nodes. Nodes in a Brokerless system may act as both Broker and client. A node that supports Broker functionality may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and Broker nodes in a system with full interoperability.

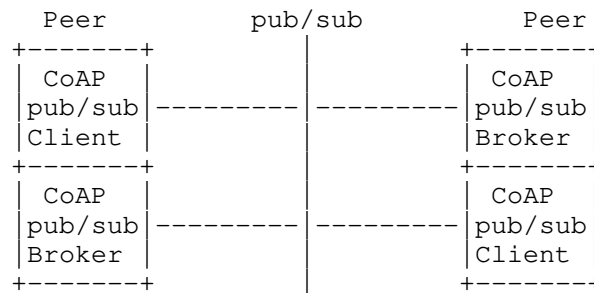


Figure 2: Brokerless pub/sub

4. CoAP Pub/sub REST API

This section defines the REST API exposed by a CoAP pub/sub Broker to pub/sub Clients. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pub/sub Broker implementing this specification SHOULD support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. Optimized implementations MAY support a subset of the operations as required by particular constrained use cases.

4.1. DISCOVERY

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to the entry point of its pub/sub API at its .well-known/core location [RFC6690]. A CoAP pub/sub Broker MAY register its pub/sub REST API entry point with a Resource Directory. Figure 3 shows an example of a client discovering a local pub/sub API using CoAP Simple Discovery. A Broker wishing to advertise the CoAP pub/sub API for Simple Discovery or through a Resource Directory MUST use the link relation `rt=core.ps`. A Broker MAY advertise its supported content formats and other attributes in the link to its pub/sub API.

A CoAP pub/sub Broker MAY offer a topic discovery entry point to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (`rt`) of "temperature" using Discover. The client then receives the URI of the resource and its content-format. A pub/sub Broker wishing to advertise topic discovery MUST use the relation `rt=core.ps.discover` in the link.

A CoAP pub/sub Broker MAY provide topic discovery functionality through the .well-known/core resource. Links to topics may be exposed at .well-known/core in addition to links to the pub/sub API. Figure 5 shows an example of topic discovery through .well-known/core.

Topics in the broker may be created in hierarchies (see Section 4.2) with parent topics having sub-topics. For a discovery the broker may choose to not expose the sub-topics in order to limit amount of topic links sent in a discovery response. The client can then perform discovery for the parent topics it wants to discover the sub-topics.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: application/link-format

The following response codes are defined for the DISCOVER operation:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the Broker resource. A pub/sub Broker SHOULD use the value "/ps/" for the base URI of the pub/sub API wherever possible.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

Client	Broker
----- GET /.well-known/core?rt=core.ps ----->> -- Content-Format: application/link-format --- <<---- 2.05 Content </ps/>;rt=core.ps;rt=core.ps.discover;ct=40 --	

Figure 3: Example of DISCOVER pub/sub function

Client	Broker
<pre> ----- GET /ps/?rt="temperature" ----->> Content-Format: application/link-format <<-- 2.05 Content </ps/currentTemp>;rt="temperature";ct=50 --- </pre>	

Figure 4: Example of DISCOVER topic

Client	Broker
<pre> ----- GET /.well-known/core?ct=50 ----->> Content-Format: application/link-format <<-- 2.05 Content </ps/currentTemp>;rt="temperature";ct=50 --- </pre>	

Figure 5: Example of DISCOVER topic

4.2. CREATE

A CoAP pub/sub broker SHOULD allow Clients to create new topics on the broker using CREATE. Some exceptions are for fixed brokerless devices and pre-configured brokers in dedicated installations. A client wishing to create a topic MUST use a CoAP POST to the pub/sub API with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. Additional link target attributes and relation values MAY be included in the topic specification link when a topic is created.

The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request.

Topic hierarchies can be created by creating parent topics. A parent topic is created with a POST using ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690], such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link

serialization, that topic may then have sub-topics created under it as shown in Figure 7.

Only one level in the topic hierarchy may be created as a result of a CREATE operation, unless create on PUBLISH is supported (see Section 4.3). The topic string used in the link target MUST NOT contain the "/" character.

A topic creator MUST include exactly one content format link attribute value (ct) in the create payload. If the content format option is not included or if the option is repeated, the Broker MUST reject the operation with an error code of "4.00 Bad Request".

Only one topic may be created per request. If there is more than one link included in a CREATE request, the Broker MUST reject the operation with an error code of "4.00 Bad Request".

A Broker MUST return a response code of "2.01 Created" if the topic is created and MUST return the URI path of the created topic via Location-Path options. If a new topic can not be created, the Broker MUST return the appropriate 4.xx response code indicating the reason for failure.

A Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. A Broker SHOULD retain a topic indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

A topic creator SHOULD PUBLISH an initial value to a newly-created Topic in order to enable responses to READ and SUBSCRIBE requests that may be submitted after the topic is discoverable.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for the CREATE operation:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Figure 6 shows an example of a topic called "topic1" being successfully created.

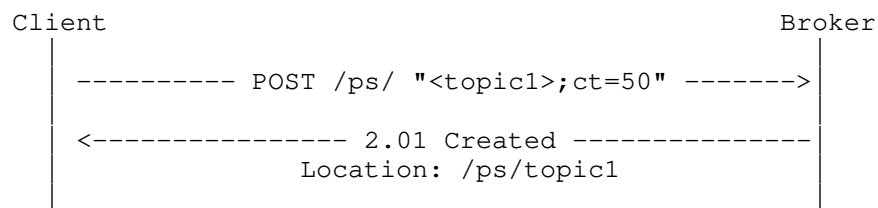


Figure 6: Example of CREATE topic

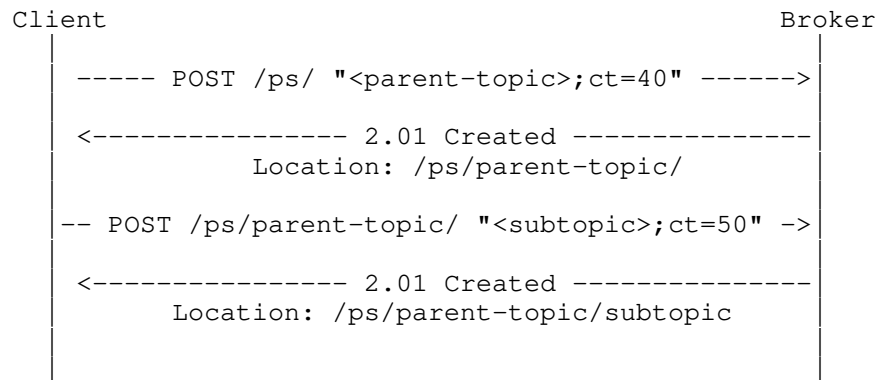


Figure 7: Example of CREATE of topic hierarchy

4.3. PUBLISH

A CoAP pub/sub Broker MAY allow clients to PUBLISH to topics on the Broker. A client MAY use the PUT or the POST method to publish state updates to the CoAP pub/sub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The Broker MUST reject publish operations which do not

use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The Broker MUST return a response code of "2.04 Changed" if the publish is accepted. A Broker MAY return a "4.04 Not Found" if the topic does not exist. A Broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

A Broker MUST accept PUBLISH operations using the PUT method. PUBLISH operations using the PUT method replace any stored representation associated with the topic, with the supplied representation. A Broker MAY reject, or delay responses to, PUT requests to a topic while pending resolution of notifications to subscribers from previous PUT requests.

Create on PUBLISH: A Broker MAY accept PUBLISH operations to new topics using the PUT method. If a Broker accepts a PUBLISH using PUT to a topic that does not exist, the Broker MUST create the topic using the information in the PUT operation. The Broker MUST create a topic with the URI-Path of the request, including all of the sub-topics necessary, and create a topic link with the ct attribute set to the content-format value from the header of the PUT request. If topic is created, the Broker MUST return the response "2.01 Created" with the URI of the created topic, including all of the created path segments, returned via the Location-Path option.

Figure 9 shows an example of a topic being created on first PUBLISH.

A Broker MAY accept PUBLISH operations using the POST method. If a Broker accepts PUBLISH using POST it shall respond with the 2.04 Changed status code. If an attempt is made to PUBLISH using POST to a topic that does not exist, the Broker SHALL return a response status indicating resource not found, such as HTTP 404 or CoAP 4.04.

A Broker MAY perform garbage collection of stored representations which have been delivered to all subscribers or which have timed out. A Broker MAY retain at least one most recently published representation to return in response to SUBSCRIBE and READ requests.

A Broker MUST make a best-effort attempt to notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 10.

If a client publishes to a Broker without the Max-Age option, the Broker MUST refresh the topic lifetime with the most recently set Max-Age value, and the Broker MUST include the most recently set Max-Age value in the Max-Age option of all notifications.

If a client publishes to a Broker with the Max-Age option, the Broker MUST include the same value for the Max-Age option in all notifications.

A Broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH operation is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for the PUBLISH operation:

Success: 2.01 "Created". Successful publish, topic is created

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 10 for an example of a Broker forwarding a message from a publishing client to a subscribed client.

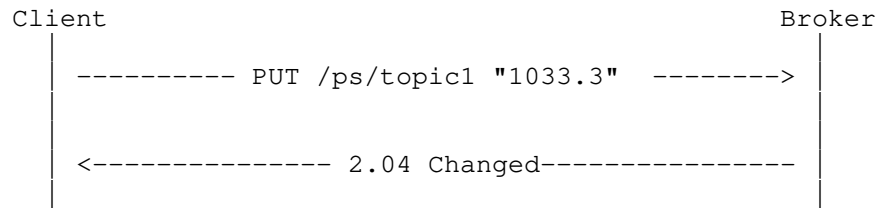


Figure 8: Example of PUBLISH

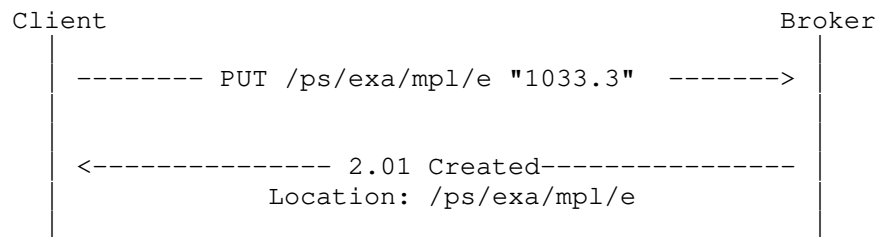


Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

A CoAP pub/sub Broker MAY allow Clients to subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pub/sub Client wishing to Subscribe to a topic on a Broker MUST use a CoAP GET with the Observe option set to 0 (zero). The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the Broker can supply the requested content format. The Broker MUST reject Subscribe requests on a topic if the content format of the request is not the content format the topic was created with.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed, or if Max-Age of a previously published representation has expired.

If a Topic has been created but not yet published to when a SUBSCRIBE to the topic is received, the Broker MAY acknowledge and queue the

pending SUBSCRIBE and defer the response until an initial PUBLISH occurs.

The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per [RFC7641] Section 4.1.

There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

The following response codes are defined for the SUBSCRIBE operation:

Success: 2.05 "Content". Successful subscribe, current value included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the Broker, with a subsequent notification. The subscribe response from the Broker uses the last stored value

associated with the topic1. The notification from the Broker is sent in response to the publish received from Client1.

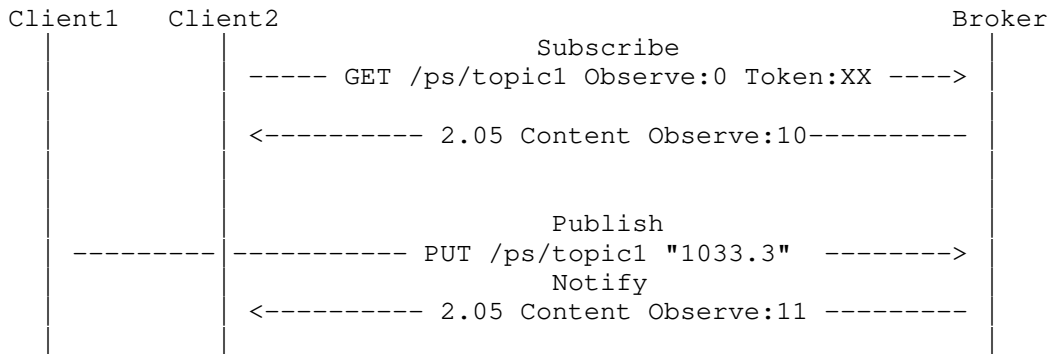


Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

If a CoAP pub/sub Broker allows clients to SUBSCRIBE to topics on the Broker, it MUST allow Clients to unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pub/sub Client wishing to unsubscribe to a topic on a Broker MUST either use CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

The following response codes are defined for the UNSUBSCRIBE operation:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.07 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 11 shows an example of a client unsubscribe using the Observe=1 cancellation method.

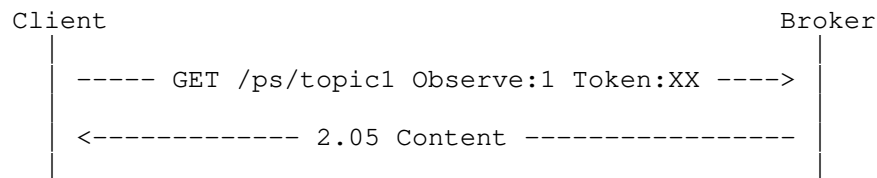


Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub Broker MAY accept Read requests on a topic using the the CoAP GET method if the content format of the request matches the content format the topic was created with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the Broker can supply the requested content format.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed, or if Max-Age of a previously published representation has expired.

If a Topic has been created but not yet published to when a READ to the topic is received, the Broker MAY acknowledge and queue the pending READ, and defer the response until an initial PUBLISH occurs.

The Broker MUST return a response code "4.15 Unsupported Content Format" if the Broker can not return the requested content format.

The READ operation is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

The following response codes are defined for the READ operation:

Success: 2.05 "Content". Successful READ, current value included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 12 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

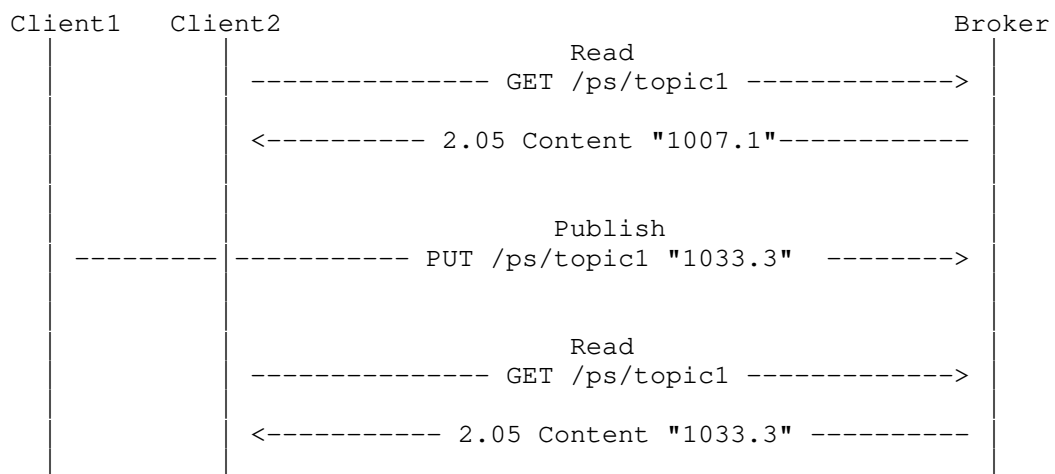


Figure 12: Example of READ

4.7. REMOVE

A CoAP pub/sub Broker MAY allow clients to remove topics from the Broker using the CoAP Delete method on the URI of the topic. The CoAP pub/sub Broker MUST return "2.02 Deleted" if the removal is successful. The Broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed.

When a topic is removed for any reason, the Broker SHOULD remove all of the observers from the list of observers and return a final 4.04 "Not Found" response as per [RFC7641] Section 3.2. If a topic which has sub-topics is removed, then all of its sub-topics MUST be recursively removed.

The REMOVE operation is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: None

Response Payload: None

The following response codes are defined for the REMOVE operation:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 13 shows a successful remove of topic1.

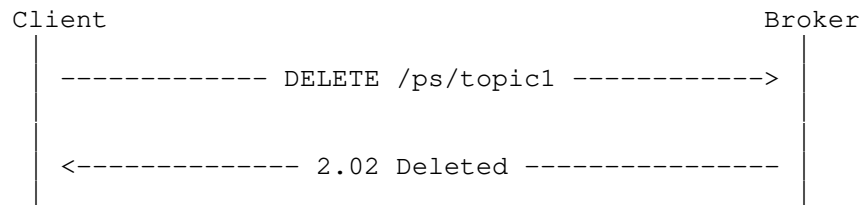


Figure 13: Example of REMOVE

5. CoAP Pub/sub Operation with Resource Directory

A CoAP pub/sub Broker may register the base URI, which is the REST API entry point for a pub/sub service, with a Resource Directory. A pub/sub Client may use an RD to discover a pub/sub Broker.

A CoAP pub/sub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pub/sub topics. A pub/sub Client may use an RD to discover pub/sub Topics. A client which registers pub/sub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pub/sub Broker.

A CoAP pub/sub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pub/sub Broker. See Section 4.1 in this document.

The pub/sub Broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource

Directory. The RD server will then retrieve the links from the .well-known/core of the pub/sub Broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pub/sub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the Broker, allowing the Broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the Broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the Broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the Broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the Broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the Broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the Broker is unable to serve a certain client that is sending publish messages too fast, the Broker SHOULD respond with Response Code 4.29, "Too Many Requests" [RFC8516] and set the Max-Age Option to indicate the number of seconds after which the client can retry. The Broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the Broker for a publish message to a topic, it MUST NOT send new publish messages to the Broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pub/sub Broker and the

clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the Broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pub/sub Broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since Brokers terminate the exchange. While running separate DTLS sessions from the client device to the Broker and from Broker to client application protects confidentiality on those paths, the client device does not know whether the commands coming from the Broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected data can also be verified by the intermediate Broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The Broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pub/sub Broker, the use of end-to-end object security is RECOMMENDED as described in [I-D.palombini-ace-coap-pubsub-profile]. An example application that uses the CoAP pub/sub Broker and relies on end-to-end object security is described in [RFC8387]. When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the Broker and set the option Content-Format to application/smpayl. Upon receipt, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- o Attribute Value: core.ps
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.2. Resource Type value 'core.ps.discover'

- o Attribute Value: core.ps.discover
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.

- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8516] Keranen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.

11.2. Informative References

- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.
- [I-D.palombini-ace-coap-pubsub-profile]
Palombini, F., "CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-palombini-ace-coap-pubsub-profile-05 (work in progress), July 2019.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<https://www.rfc-editor.org/info/rfc5988>>.

[RFC8387] Sethi, M., Arkko, J., Keranen, A., and H. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", RFC 8387, DOI 10.17487/RFC8387, May 2018, <<https://www.rfc-editor.org/info/rfc8387>>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smarththings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: March 30, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
I. Petrov, Ed.
Acklio
September 27, 2019

CoAP Management Interface
draft-ietf-core-comi-08

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access datastore and data node resources specified in YANG, or SMIV2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. The complete solution composed of CoMI, [I-D.ietf-core-yang-cbor] and [I-D.ietf-core-sid] is called CORECONF. CORECONF extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

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

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 March 30, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. CoMI Architecture	5
2.1. Major differences between RESTCONF and CORECONF	6
2.1.1. Differences due to CoAP and its efficient usage	6
2.1.2. Differences due to the use of CBOR	7
2.2. Compression of YANG identifiers	7
2.3. Instance-identifier	8
2.4. Content-Formats	8
2.5. Unified datastore	10
3. Example syntax	11
4. CoAP Interface	11
4.1. Using the 'k' Uri-Query option	13
4.2. Data Retrieval	14
4.2.1. Using the 'c' Uri-Query option	14
4.2.2. Using the 'd' Uri-Query option	15
4.2.3. GET	16
4.2.4. FETCH	18
4.3. Data Editing	19
4.3.1. Data Ordering	19
4.3.2. POST	20
4.3.3. PUT	21
4.3.4. iPATCH	21
4.3.5. DELETE	22
4.4. Full datastore access	23
4.4.1. Full datastore examples	24

4.5.	Event stream	24
4.5.1.	Notify Examples	25
4.5.2.	The 'f' Uri-Query option	26
4.6.	RPC statements	27
4.6.1.	RPC Example	27
5.	Use of Block-wise Transfers	29
6.	Application Discovery	29
6.1.	YANG library	29
6.2.	Resource Discovery	30
6.2.1.	Datastore Resource Discovery	30
6.2.2.	Data node Resource Discovery	31
6.2.3.	Event stream Resource Discovery	31
7.	Error Handling	31
8.	Security Considerations	35
9.	IANA Considerations	35
9.1.	Resource Type (rt=) Link Target Attribute Values Registry	35
9.2.	CoAP Content-Formats Registry	36
9.3.	Media Types Registry	36
10.	Acknowledgements	38
11.	References	38
11.1.	Normative References	38
11.2.	Informative References	40
Appendix A.	ietf-comi YANG module	40
Appendix B.	ietf-comi .sid file	46
Authors'	Addresses	50

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 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 [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

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.

The following terms are defined in the YANG data modelling 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 (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 [I-D.ietf-core-yang-cbor].

2. CoMI Architecture

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

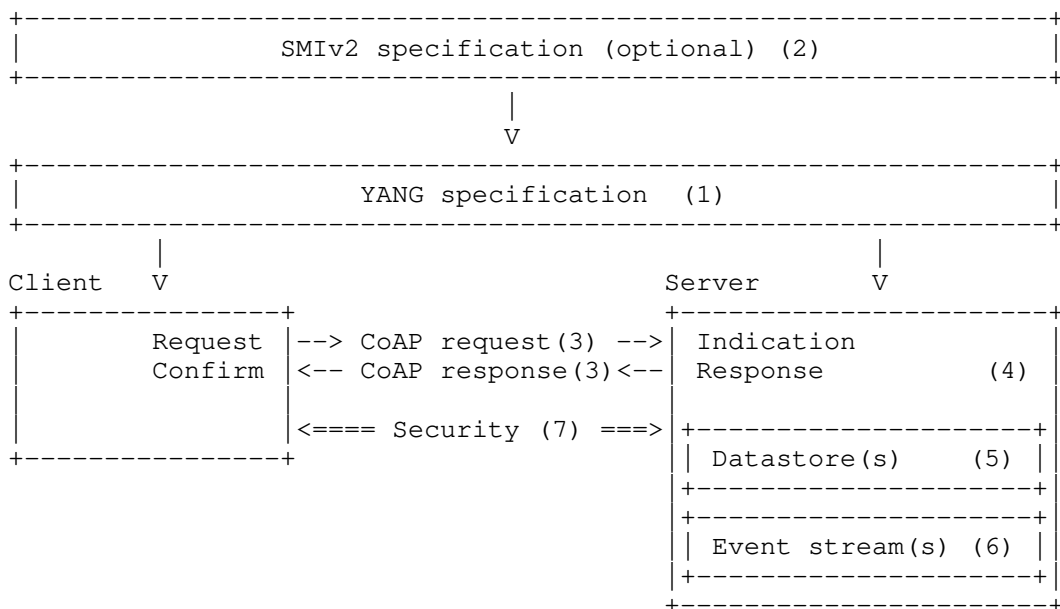


Figure 1: Abstract CoMI architecture

Figure 1 is a high-level representation of the main elements of the CoMI management architecture. The different numbered components of Figure 1 are discussed according to 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 CoMI client sends request messages to and receives response messages from the CoMI server.
- (4) Request, Indication, Response, Confirm: Processes performed by the CoMI clients and servers.
- (5) Datastore: A resource used to access configuration data, state data, RPCs and actions. A CoMI 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 CoMI 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 CoMI resources. CoMI relies on security protocols such as DTLS [RFC6347] or OSCOAP [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 counts. 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

- o CORECONF uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats.
- o 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.

- o RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not supported by CoAP.
- o CORECONF does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.
- o 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

- o CORECONF encodes YANG identifier strings as numbers, where RESTCONF does not.
- o CORECONF also differ 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 CoMI, numeric identifiers called YANG Schema Item Identifier (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 [RFC4648] section 5, 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 (value 0) at the start of the resulting string are removed. See Figure 2 for complete illustration.

```
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 follow.

```
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 "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 [RFC7950] section 9.13. 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 [I-D.ietf-core-yang-cbor] section 6.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' URI-Query as defined in Section 4.1.

2.4. Content-Formats

CoMI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor].

The following Content-formats are defined:

application/yang-data+cbor: This Content-Format represents a CBOR YANG document containing one or multiple data node values. Each data node is identified by its associated SID.

FORMAT: CBOR map of SID, instance-value

The message payload of Content-Format 'application/yang-data+cbor' is encoded using a CBOR map. Each entry within the CBOR map contains the data node identifier (i.e. SID) and the associated instance-value. Instance-values are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 4.

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

FORMAT: CBOR array of instance-identifier

The message payload of Content-Format '`application/yang-identifiers+cbor`' is encoded using a CBOR array. Each entry of this CBOR array contain an instance-identifier encoded as defined in [I-D.ietf-core-yang-cbor] section 6.13.1.

`application/yang-instances+cbor`: This Content-Format 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 array of CBOR map of instance-identifier, instance-value

The message payload of Content-Format '`application/yang-instances+cbor`' is encoded using a CBOR array. Each entry within this CBOR array contains a CBOR map carrying an instance-identifier and associated instance-value. Instance-identifiers are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1, instance-values are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 4.

When present in an iPATCH request payload, this Content-Format 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 Content-format usages are summarized in the table below:

Method	Resource	Content-Format
GET response	data node	/application/yang-data+cbor
PUT request	data node	/application/yang-data+cbor
POST request	data node	/application/yang-data+cbor
DELETE	data node	n/a
GET response	datastore	/application/yang-data+cbor
PUT request	datastore	/application/yang-data+cbor
POST request	datastore	/application/yang-data+cbor
FETCH request	datastore	/application/yang-identifiers+cbor
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
POST response	rpc, action	/application/yang-data+cbor

2.5. Unified datastore

CoMI 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, CoMI 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-comi"

3. Example syntax

CBOR is used to encode CoMI request and response payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

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

4. CoAP Interface

This note specifies a Management Interface. CoAP endpoints that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.ds, with example path: /c, where c is short-hand for CoMI/CORECONF. The path /c is recommended, but not compulsory (see Section 6).

The mapping of YANG data node instances to CoMI resources is as follows. Every data node of the YANG modules loaded in the CoMI server represents a sub-resource of the datastore resource (e.g. /c/sid). 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.

CoMI 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 CoMI management interface is shown in the table below:

CoAP resource	Recommended path	rt
Datastore resource	/c	core.c.ds
Data node resource	/c/SID	core.c.dn
Event stream resource	/s	core.c.ev

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

The methods used by CoMI 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	Idem-potently create, replace, and delete data node resource(s) within a datastore resource
DELETE	Delete a datastore resource or a data node resource

There is one Uri-Query option for the GET, PUT, POST, and DELETE methods.

Uri-Query option	Description
k	Select an instance within YANG list(s)

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

4.1. Using the 'k' Uri-Query option

The "k" (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1,key2,...). Where SID identifies a data node, and key1, key2 are 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:

- o For a given list, if a parent data node is a list, generate the keys for the parent list first.
- o For a given list, generate key values in the order specified in the YANG module.

Key values are encoded using the rules defined in the following table.

YANG datatype	Uri-Query text content
uint8,uint16,uint32, uint64	int2str(key)
int8, int16,int32, int64	urlSafeBase64(CBORencode(key))
decimal64	urlSafeBase64(CBOR key)
string	key
boolean	"0" or "1"
enumeration	int2str(key)
bits	urlSafeBase64(CBORencode(key))
binary	urlSafeBase64(key)
identityref	int2str(key)
union	urlSafeBase64(CBORencode(key))
instance-identifier	urlSafeBase64(CBORencode(key))

In this table:

- o The method `int2str()` is used to convert an integer value to a decimal string. For example, `int2str(0x0123)` return the string "291".
- o The method `urlSafeBase64()` is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. For example, `urlSafeBase64(\xF9\x56\xA1\x3C)` return the string "-VahPA".
- o The method `CBORencode()` is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 6.

The resulting key string is encoded in a Uri-Query as specified in [RFC7252] section 6.5.

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 Uri-Query options for the GET and FETCH methods.

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

4.2.1. Using the 'c' Uri-Query option

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

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 Uri-Query option is not present, the default value is "a".

4.2.2. Using the 'd' Uri-Query option

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].

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 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 Uri-Query option is not present, the default value is 't'.

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 'k' Uri-Query option is added if the data node is an instance of a list.

FORMAT:

GET <data node resource> ['k' Uri-Query option]

2.05 Content (Content-Format: application/yang-data+cbor)
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 [I-D.ietf-core-yang-cbor] section 6.4.

REQ: GET example.com/c/a7

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

The next example represents the retrieval of a YANG container. In this case, the CoMI 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 [I-D.ietf-core-yang-cbor] section 4.2.

```

REQ: GET example.com/c/a5

RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
  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 using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

```

REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
  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 instance within the /interfaces/interface YANG list, the CoMI client adds the key of the targeted instance in its CoAP request using the 'k' URI-Query. The return payload containing the instance requested is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing the requested instance.

REQ: GET example.com/c/X9?k="eth0"

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

```
{
  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 instance 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 [I-D.ietf-core-yang-cbor] section 6.4.

REQ: GET example.com/c/X-?k="eth0"

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

```
{
  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-identifier 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 implementations complexity since the format of each entry within the CBOR array 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 array of instance-identifier

  2.05 Content (Content-Format: application/yang-instances+cbor)
  CBOR array of CBOR map 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` [RFC7223]. 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) /
    }
  }
]

```

4.3. Data Editing

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

4.3.1. Data Ordering

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

request. 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 CoMI for 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)
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 [RFC7223]. This example creates a new list instance within the interface list (SID = 1533):

```
REQ: POST /c/X9 (Content-Format: application/yang-data+cbor)
{
  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> ['k' Uri-Query option]
      (Content-Format: application/yang-data+cbor)
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 [RFC7223]. This example updates the instance of the list interface (SID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0" (Content-Format: application/yang-data+cbor)
{
  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 Uri-Query options for the iPATCH method.

The processing of the iPATCH command is specified by Content-Format '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 array of CBOR map of instance-identifier, instance-value

2.04 Changed

4.3.4.1. iPATCH example

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

- o Set "/system/ntp/enabled" (SID 1755) to true.
- o Remove the server "tac.nrc.ca" from the "/system/ntp/server" (SID 1756) list.
- o 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 : {
      3 : "tic.nrc.ca",        / name (SID 1759) /
      4 : true,                / prefer (SID 1760) /
      5 : {
        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> ['k' Uri-Query option]

2.02 Deleted

4.3.5.1. DELETE example

This example uses the interface list from module ietf-interfaces [RFC7223]. This example deletes an instance of the interface list (SID = 1533):

REQ: DELETE /c/X9?k="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)
CBOR map of SID, instance-value

FORMAT:

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

2.04 Changed

FORMAT:

POST <datastore resource>
(Content-Format: application/yang-data+cbor)
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 [RFC7223] 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)

```
{
  1721 : {
    2: "2016-10-26T12:16:31Z", / current-datetime (SID 1723) /
    1: "2014-10-05T09:00:00Z" / boot-datetime (SID 1722) /
  },
  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) /
    }
  ]
}
```

4.5. Event stream

Event notification is an essential function for the management of servers. CoMI allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The recommended path of the default event stream is /s. 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, specifying the /s resource when the default stream is selected.

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

The format of notification without any content is a null value. The format of single notification is defined in [I-D.ietf-core-yang-cbor] section 4.2.1. For multiple notifications the format is an array where each element is a single notification as described in [I-D.ietf-core-yang-cbor] section 4.2.1.

FORMAT:

GET <stream-resource> Observe(0)

2.05 Content (Content-Format: application/yang-instances+cbor)
CBOR array of CBOR map of instance-identifier, instance-value

The array of data node instances may contain identical entries 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;
    }
  }
}
```

By executing a GET on the /s resource the client receives the following response:

REQ: GET /s Observe(0)

```
RES: 2.05 Content (Content-Format: application/yang-tree+cbor)
      Observe(12)
[
  {
    60010 : {
      1 : "0/4/21",      / example-port-fault (SID 60010) /
      2 : "Open pin 2"   / port-name (SID 60011) /
    }
  },
  {
    60010 : {
      1 : "1/4/21",      / example-port-fault (SID 60010) /
      2 : "Open pin 5"   / port-name (SID 60011) /
    }
  }
]
```

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.

To check that the client is still alive, the server MUST send Confirmable Message periodically. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

4.5.2. The 'f' Uri-Query option

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 CoMI 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 [I-D.ietf-core-yang-cbor] section 4.2.1.

The returned success response code is 2.05 Content.

FORMAT:

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

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

4.6.1. RPC Example

The example is based on the YANG action "reset" as defined in [RFC7950] section 7.15.3 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?k="myserver"
      (Content-Format: application/yang-data+cbor)
{
  60002 : {
    1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
  }
}

RES:  2.05 Content (Content-Format: application/yang-data+cbor)
{
  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. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. On the server, values are changed, lists are 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.

6. Application Discovery

Two application discovery mechanisms are supported by CoMI, 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 CoMI server. From this information, CoMI 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.yml"`. 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 encoded as specified in Section 2.2).

```
REQ: GET /.well-known/core?rt=core.c.yml
```

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

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 CoMI 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
```

For example:

```
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
```


6.2.2. Data node Resource Discovery

If implemented, the presence and location of (path to) each data node implemented by the CoMI 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.

```
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 CoMI 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.

For example:

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

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

7. Error Handling

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

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

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

- o Error 4.01 (Unauthorized) is returned by the CoMI server when the CoMI client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.02 (Bad Option) is returned by the CoMI server when one or more CoAP options are unknown or malformed.
- o Error 4.04 (Not Found) is returned by the CoMI server when the CoMI client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.05 (Method Not Allowed) is returned by the CoMI server when the CoMI 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).
- o Error 4.08 (Request Entity Incomplete) is returned by the CoMI server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.
- o Error 4.13 (Request Entity Too Large) may be returned by the CoMI server during a block transfer request, see [RFC7959] for more details.
- o Error 4.15 (Unsupported Content-Format) is returned by the CoMI server when the Content-Format used in the request does not match those specified in section Section 2.4.

The CoMI server MUST also enforce the different constraints associated to the YANG data models implemented. These constraints are described in [RFC7950] section 8. 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 [I-D.ietf-core-yang-cbor] section 5.

```
+--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-comi YANG module, these tags are implemented as YANG identity and can be extended as needed.

- o error-tag 'operation-failed' is returned by the CoMI server when the operation request cannot be processed successfully.
 - * error-app-tag 'malformed-message' is returned by the CoMI server when the payload received from the CoMI client does not contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or does not comply with the CBOR structure defined within this document.
 - * error-app-tag 'data-not-unique' is returned by the CoMI 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 CoMI 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 CoMI 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 CoMI server when the restrictions imposed by a 'must' statement are violated.
 - * error-app-tag 'duplicate' is returned by the CoMI server when a client tries to create a duplicate list or leaf-list entry.
- o error-tag 'invalid-value' is returned by the CoMI server when the CoMI 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 CoMI 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 CoMI server when the validation of the 'range' property fails.
- * error-app-tag 'invalid-length' is returned by the CoMI server when the validation of the 'length' property fails.
- * error-app-tag 'pattern-test-failed' is returned by the CoMI server when the validation of the 'pattern' property fails.
- o error-tag 'missing-element' is returned by the CoMI server when the operation requested by a CoMI 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 CoMI server to further qualify a missing-element error. This error is returned when the CoMI client tries to create or list instance, without all the 'key' specified or when the CoMI client tries to delete a leaf listed as a 'key'.
- * error-app-tag 'missing-input-parameter' is returned by the CoMI server when the input parameters of an RPC or action are incomplete.
- o error-tag 'unknown-element' is returned by the CoMI server when the CoMI client tries to access a data node of a YANG module not supported, of a data node associated to an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.
- o error-tag 'bad-element' is returned by the CoMI server when the CoMI client tries to create data nodes for more than one case in a choice.
- o error-tag 'data-missing' is returned by the CoMI server when a data node required to accept the request is not present.
- * error-app-tag 'instance-required' is returned by the CoMI 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 CoMI server when no nodes exist in a mandatory choice.

- o error-tag 'error' is returned by the CoMI server when an unspecified error has occurred.

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

```
RES: 4.00 Bad Request (Content-Format: application/yang-data+cbor)
{
  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. CoMI re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]).

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

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

// 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-data+cbor		TBD1	RFC XXXX
application/yang-identifiers+cbor		TBD2	RFC XXXX
application/yang-instances+cbor		TBD3	RFC XXXX

// 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-data+cbor	application/yang-data+cbor	RFC XXXX
yang-identifiers+cbor	application/ yang-identifiers+cbor	RFC XXXX
yang-instances+cbor	application/ yang-instances+cbor	RFC XXXX

Each of these media types share the following information:

- o Subtype name: <as listed in table>
- o Required parameters: N/A
- o Optional parameters: N/A
- o Encoding considerations: binary
- o Security considerations: See the Security Considerations section of RFC XXXX
- o Interoperability considerations: N/A
- o Published specification: RFC XXXX
- o Applications that use this media type: CoMI
- o Fragment identifier considerations: N/A
- o 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
- o Person & email address to contact for further information:
iesg&ietf.org

- o Intended usage: COMMON
- o Restrictions on usage: N/A
- o Author: Michel Veillette, ietf@augustcellars.com
- o Change Controller: IESG
- o Provisional registration? No

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

10. Acknowledgements

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

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

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

11. References

11.1. Normative References

- [I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-11 (work in progress), September 2019.
- [I-D.ietf-core-yang-library]
Veillette, M. and I. Petrov, "Constrained YANG Module Library", draft-ietf-core-yang-library-00 (work in progress), July 2019.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/info/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/info/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/info/rfc6243>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/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/info/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/info/rfc7959>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [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/info/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/info/rfc8174>>.

11.2. 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/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [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/info/rfc8342>>.
- [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/info/rfc8613>>.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2019-03-28.yang"
module ietf-comi {
  yang-version 1.1;

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

  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 CoMI 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 2019-03-28 {
  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 CoMI server when the operation request
    can't be processed successfully.";
}

identity invalid-value {
  base error-tag;
  description
    "Returned by the CoMI server when the CoMI 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 CoMI server when the operation requested
    by a CoMI 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 CoMI server when the CoMI 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 CoMI server when the CoMI client tries to
            create data nodes for more than one case in a choice.";
    }

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

    identity error {
        base error-tag;
        description
            "Returned by the CoMI 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 CoMI server when the payload received
            from the CoMI client don't contain a well-formed CBOR
            content as defined in [RFC7049] section 3.3 or don't
            comply with the CBOR structure defined within this
            document.";
    }

    identity data-not-unique {
        base error-app-tag;
        description
            "Returned by the CoMI 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 CoMI 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 CoMI 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 CoMI server when the restrictions
       imposed by a 'must' statement are violated.";
  }

  identity duplicate {
    base error-app-tag;
    description
      "Returned by the CoMI 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 CoMI server when CBOR encoding is
       incorrect 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 CoMI server when the validation of the
       'range' property fails.";
  }

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

```
}

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

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

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

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

rc:yang-data comi-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-comi .sid file

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-comi",
  "module-revision": "2019-03-28",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-comi",

```



```
    "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-comi:error",
      "sid": 1024
    },
    {
      "namespace": "data",
      "identifier": "/ietf-comi:error/error-app-tag",
      "sid": 1025
    },
    {
      "namespace": "data",
      "identifier": "/ietf-comi:error/error-data-node",
      "sid": 1026
    },
    {
      "namespace": "data",
      "identifier": "/ietf-comi:error/error-message",
      "sid": 1027
    },
    {
      "namespace": "data",
```

```
    "identifiant": "/ietf-comi:error/error-tag",  
    "sid": 1028  
  }  
]  
}
```

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: consultancy@vanderstok.org
URI: www.vanderstok.org

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

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

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

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

K. Hartke
Ericsson
November 4, 2019

The Constrained RESTful Application Language (CoRAL)
draft-ietf-core-coral-01

Abstract

The Constrained RESTful Application Language (CoRAL) defines a data model and interaction model as well as two specialized serialization formats for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), as well as simple resource metadata.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Notational Conventions	4
2.	Data and Interaction Model	4
2.1.	Browsing Context	5
2.2.	Documents	5
2.3.	Links	5
2.4.	Forms	6
2.5.	Form Fields	7
2.6.	Embedded Representations	7
2.7.	Navigation	8
2.8.	History Traversal	9
3.	Binary Format	10
3.1.	Data Structure	10
3.1.1.	Documents	10
3.1.2.	Links	10
3.1.3.	Forms	11
3.1.4.	Embedded Representations	12
3.1.5.	Directives	13
3.2.	Dictionaries	13
3.2.1.	Dictionary References	13
3.2.2.	Media Type Parameter	14
4.	Textual Format	14
4.1.	Lexical Structure	15
4.1.1.	Line Terminators	15
4.1.2.	White Space	15
4.1.3.	Comments	15
4.1.4.	Identifiers	16
4.1.5.	IRIs and IRI References	16
4.1.6.	Literals	16
4.1.7.	Punctuators	20
4.2.	Syntactic Structure	20
4.2.1.	Documents	21
4.2.2.	Links	21
4.2.3.	Forms	22
4.2.4.	Embedded Representations	23
4.2.5.	Directives	23
5.	Usage Considerations	24
5.1.	Specifying CoRAL-based Applications	24
5.1.1.	Application Interfaces	25
5.1.2.	Resource Identifiers	25
5.1.3.	Implementation Limits	26
5.2.	Minting Vocabulary	26
5.3.	Expressing Registered Link Relation Types	27
5.4.	Expressing Simple RDF Statements	28
5.5.	Expressing Natural Language Texts	28
5.6.	Embedding CoRAL in CBOR Data	29

5.7.	Submitting CoRAL Documents	29
5.7.1.	PUT Requests	29
5.7.2.	POST Requests	29
5.8.	Returning CoRAL Documents	30
5.8.1.	Success Responses	30
5.8.2.	Error Responses	30
6.	Security Considerations	30
7.	IANA Considerations	32
7.1.	Media Type "application/coral+cbor"	32
7.2.	Media Type "text/coral"	33
7.3.	CoAP Content Formats	34
7.4.	CBOR Tag	35
8.	References	35
8.1.	Normative References	35
8.2.	Informative References	37
Appendix A.	Core Vocabulary	39
A.1.	Base	40
A.2.	Collections	41
A.3.	HTTP	41
A.4.	CoAP	42
Appendix B.	Default Dictionary	43
Acknowledgements	44
Author's Address	44

1. Introduction

The Constrained RESTful Application Language (CoRAL) is a language for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), as well as simple resource metadata.

CoRAL is intended for driving automated software agents that navigate a Web application based on a standardized vocabulary of link relation types and operation types. It is designed to be used in conjunction with a Web transfer protocol such as the Hypertext Transfer Protocol (HTTP) [RFC7230] or the Constrained Application Protocol (CoAP) [RFC7252].

This document defines the CoRAL data and interaction model, as well as two specialized CoRAL serialization formats.

The CoRAL data and interaction model is a superset of the Web Linking model of RFC 8288 [RFC8288]. The data model consists of two primary elements: "links" that describe the relationship between two resources and the type of that relationship, and "forms" that describe a possible operation on a resource and the type of that operation. Additionally, the data model can describe simple resource metadata in a way similar to the Resource Description Framework (RDF)

[W3C.REC-rdf11-concepts-20140225]. In contrast to RDF, the focus of CoRAL however is on the interaction with resources, not just the relationships between them. The interaction model derives from HTML 5 [W3C.REC-html52-20171214] and specifies how an automated software agent can navigate between resources by following links and perform operations on resources by submitting forms.

The primary CoRAL serialization format is a compact, binary encoding of links and forms in Concise Binary Object Representation (CBOR) [RFC7049]. It is intended for environments with constraints on power, memory, and processing resources [RFC7228] and shares many similarities with the message format of the Constrained Application Protocol (CoAP) [RFC7252]: For example, it uses numeric identifiers instead of verbose strings for link relation types and operation types, and pre-parses Uniform Resource Identifiers (URIs) [RFC3986] into (what CoAP considers to be) their components, which simplifies URI processing for constrained nodes a lot. As a result, link serializations in CoRAL are often much more compact than equivalent serializations in CoRE Link Format [RFC6690].

The secondary CoRAL serialization format is a lightweight, textual encoding of links and forms that is intended to be easy to read and write for humans. The format is loosely inspired by the syntax of Turtle [W3C.REC-turtle-20140225] and is mainly intended for giving examples.

1.1. Notational Conventions

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.

Terms defined in this document appear in *_cursive_* where they are introduced.

2. Data and Interaction Model

The Constrained RESTful Application Language (CoRAL) is designed for building Web-based applications [W3C.REC-webarch-20041215] in which automated software agents navigate between resources by following links and perform operations on resources by submitting forms.

2.1. Browsing Context

Borrowing from HTML 5 [W3C.REC-html52-20171214], each such agent maintains a `_browsing context_` in which the representations of Web resources are processed. (In HTML 5, the browsing context typically corresponds to a tab or window in a Web browser.)

At any time, one representation in each browsing context is designated the `_active_` representation.

2.2. Documents

A resource representation in one of the CoRAL serialization formats is called a CoRAL `_document_`. The URI that was used to retrieve such a document is called the document's `_retrieval context_`.

A CoRAL document consists of a list of zero or more links, forms, and embedded resource representations, collectively called `_elements_`. CoRAL serialization formats may define additional types of elements for efficiency or convenience, such as a base for relative URI references [RFC3986].

2.3. Links

A `_link_` describes a relationship between two resources on the Web [RFC8288]. As defined in RFC 8288, it consists of a `_link context_`, a `_link relation type_`, and a `_link target_`. In CoRAL, a link can additionally have a nested list of zero or more elements, which take the place of link target attributes.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}" where the link target may be further described by nested elements.

The link relation type identifies the semantics of a link. In HTML 5 and RFC 8288, link relation types are typically denoted by an IANA-registered name, such as "stylesheet" or "type". In CoRAL, they are denoted by an Internationalized Resource Identifier (IRI) [RFC3987] such as `<http://www.iana.org/assignments/relation/stylesheet>` or `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`. This allows for the creation of new link relation types without the risk of collisions when from different organizations or domains of knowledge. An IRI also can lead to documentation, schema, and other information about the link relation type. These IRIs are only used as identity tokens, though, and are compared using Simple String Comparison (Section 5.1 of RFC 3987).

The link context and the link target are both denoted by either a URI reference or a literal (similarly to RDF). If the URI scheme indicates a Web transfer protocol such as HTTP or CoAP, then an agent can dereference the URI and navigate the browsing context to the referenced resource; this is called `_following the link_`. A literal directly identifies a value: a Boolean value, an integer, a floating-point number, a date/time value, a byte string, or a text string.

A link can occur as a top-level element in a document or as a nested element within a link. When a link occurs as a top-level element, the link context implicitly is the document's retrieval context. When a link occurs nested within a link, the link context of the inner link is the link target of the outer link.

There are no restrictions on the cardinality of links; there can be multiple links to and from a particular target, and multiple links of the same or different types between a given link context and target. However, the nested data structure constrains the description of a resource graph to a tree: Links between linked resources can only be described by further nesting links.

2.4. Forms

A `_form_` provides instructions to an agent for performing an operation on a Web resource. It consists of a `_form context_`, an `_operation type_`, a `_request method_`, and a `_submission target_`. Additionally, a form may be accompanied by a list of `_form fields_`.

A form can be viewed as an instruction of the form "To perform an {operation type} operation on {form context}, make a {request method} request to {submission target}" where the request may be further described by form fields.

The operation type identifies the semantics of the operation. Operation types are denoted like link relation types by an IRI.

The form context is the resource on which an operation is ultimately performed. To perform the operation, an agent needs to construct a request with the specified method and the specified submission target as the request URI. Usually, the submission target is the same resource as the form context, but it may be a different resource. Constructing and sending the request is called `_submitting the form_`.

Form fields, specified in the next section, can be used to provide more detailed instructions to the agent for constructing the request. For example, form fields can instruct the agent to include a payload or certain headers in the request that must match the specifications of the form fields.

A form can occur as a top-level element in a document or as a nested element within a link. When a form occurs as a top-level element, the form context implicitly is the document's retrieval context. When a form occurs nested within a link, the form context is the link target of the enclosing link.

2.5. Form Fields

Form fields provide further instructions to agents for constructing a request.

For example, a form field could identify one or more data items that need to be included in the request payload or reference another resource (such as a schema) that describes the structure of the payload. A form field could also provide other kinds of information, such as acceptable media types for the payload or expected request headers. Form fields may be specific to the protocol used for submitting the form.

A form field is the pair of a `_form field type_` and a `_form field value_`.

The form field type identifies the semantics of the form field. Form field types are denoted like link relation types and operation types by an IRI.

The form field value can be either a URI reference, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, or a text string.

2.6. Embedded Representations

When a document contains links to many resources and an agent needs a representation of each link target, it may be inefficient to retrieve each of these representations individually. To alleviate this, documents can directly embed representations of resources.

An `_embedded representation_` consists of a sequence of bytes, labeled with `_representation metadata_`.

An embedded representation may be a full, partial, or inconsistent version of the representation served from the URI of the resource.

An embedded representation can occur as a top-level element in a document or as a nested element within a link. When it occurs as a top-level element, it provides an alternate representation of the document's retrieval context. When it occurs nested within a link, it provides a representation of link target of the enclosing link.

2.7. Navigation

An agent begins interacting with an application by performing a GET request on an `_entry point URI_`. The entry point URI is the only URI an agent is expected to know before interacting with an application. From there, the agent is expected to make all requests by following links and submitting forms provided by the server in responses. The entry point URI can be obtained by manual configuration or through some discovery process.

If dereferencing the entry point URI yields a CoRAL document (or any other representation that implements the CoRAL data and interaction model), then the agent makes this document the active representation in the browsing context and proceeds as follows:

1. The first step for the agent is to decide what to do next, i.e., which type of link to follow or form to submit, based on the link relation types and operation types it understands.
2. The agent then finds the link(s) or form(s) with the respective type in the active representation. This may yield one or more candidates, from which the agent will have to select the most appropriate one. The set of candidates may be empty, for example, when a transition is not supported or not allowed.
3. The agent selects one of the candidates based on the metadata associated with each of these. Metadata includes the content type of the target resource representation, the URI scheme, the request method, and other information that is provided as nested elements in a link or form fields in a form.

If the selected candidate contains an embedded representation, the agent MAY skip the following steps and immediately proceed with step 8.

4. The agent obtains the `_request URI_` from the link target or submission target. Fragment identifiers are not part of the request URI and MUST be separated from the rest of the URI prior to a dereference.
5. The agent constructs a new request with the request URI. If the agent is following a link, then the request method MUST be GET. If the agent is submitting a form, then the request method MUST be the one specified by the form. An IRI may need to be converted to a URI (Section 3.1 of RFC 3987) for protocols that do not support IRIs.

The agent should set HTTP header fields and CoAP request options according to metadata associated with the link or form (e.g., set the HTTP Accept header field or the CoAP Accept option when the media type of the target resource is provided). Depending on the operation type of a form, the agent may also need to include a request payload that matches the specifications of one or more form fields.

6. The agent sends the request and receives the response.
7. If a fragment identifier was separated from the request URI, the agent dereferences the fragment identifier within the received representation.
8. The agent updates the browsing context by making the (embedded or received) representation the active representation.
9. Finally, the agent processes the representation according to the semantics of the content type. If the representation is a CoRAL document (or any other representation that implements the CoRAL data and interaction model), this means the agent has the choice of what to do next again -- and the cycle repeats.

2.8. History Traversal

A browsing context MAY entail a session history that lists the resource representations that the agent has processed, is processing, or will process.

An entry in the session history consists of a resource representation and the request URI that was used to retrieve the representation. New entries are added to the session history as the agent navigates from resource to resource.

An agent can navigate a browsing context by traversing the session history in addition to following links and submitting forms. For example, if an agent received a representation that doesn't contain any further links or forms, it can revert the active representation back to one it has visited earlier.

Traversing the history should take advantage of caches to avoid new requests. An agent MAY reissue a safe request (e.g., a GET request) when it doesn't have a fresh representation in its cache. An agent MUST NOT reissue an unsafe request (e.g., a PUT or POST request) unless it intends to perform that operation again.

3. Binary Format

This section defines the encoding of documents in the CoRAL binary format.

A document in the binary format is a data item in Concise Binary Object Representation (CBOR) [RFC7049]. The structure of this data item is presented in the Concise Data Definition Language (CDDL) [RFC8610]. The media type is "application/coral+cbor".

The following restrictions are placed on CBOR encoders: Byte strings and text strings MUST be encoded with definite length. Integers and floating-point values MUST be encoded as such (e.g., a floating-point value of 0.0 must not be encoded as the integer 0).

3.1. Data Structure

The data structure of a document in the binary format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) base directives. Base directives provide a way to encode URI references with a common base more efficiently.

Elements are processed in the order they appear in the document. Document processors need to maintain an `_environment_` while iterating an array of elements. The environment consists of two variables: the `_current context_` and the `_current base_`. Both the current context and the current base are initially set to the document's retrieval context.

3.1.1. Documents

The body of a document in the binary format is encoded as an array of zero or more links, forms, embedded representations, and directives.

```
document = body
```

```
body = [*(link / form / representation / directive)]
```

3.1.2. Links

A link is encoded as an array that consists of the unsigned integer 2, followed by the link relation type and the link target, optionally followed by a link body that contains nested elements.

```
link = [2, relation-type, link-target, ?body]
```

The link relation type is encoded as a text string that conforms to the syntax of an IRI [RFC3987].

relation-type = text

The link target is denoted by a Constrained Resource Identifier (CoRI) reference [I-D.ietf-core-href] or represented by a literal value. A CoRI reference MUST be resolved against the current base. The link target may be null, which indicates that the link target is an unidentified resource.

link-target = CoRI / literal

CoRI = <Defined in Section X of RFC XXXX>

literal = bool / int / float / time / bytes / text / null

The array of elements in the link body, if any, MUST be processed in a fresh environment. Both the current context and the current base in the new environment are initially set to the link target of the enclosing link.

3.1.3. Forms

A form is encoded as an array that consists of the unsigned integer 3, followed by the operation type and the submission target, optionally followed by a list of form fields.

form = [3, operation-type, submission-target, ?form-fields]

The operation type is defined in the same way as a link relation type (Section 3.1.2).

operation-type = text

The request method is either implied by the operation type or encoded as a form field. If there are both, the form field takes precedence over the operation type. Either way, the method MUST be defined for the Web transfer protocol identified by the scheme of the submission target.

The submission target is denoted by a CoRI reference. This CoRI reference MUST be resolved against the current base.

submission-target = CoRI

3.1.3.1. Form Fields

A list of form fields is encoded as an array of zero or more type-value pairs.

```
form-fields = [*(form-field-type, form-field-value)]
```

The list, if any, MUST be processed in a fresh environment. Both the current context and the current base in the new environment are initially set to the submission target of the enclosing form.

A form field type is defined in the same way as a link relation type (Section 3.1.2).

```
form-field-type = text
```

A form field value can be a CoRI reference, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, a text string, or null. A CoRI reference MUST be resolved against the current base.

```
form-field-value = CoRI / literal
```

3.1.4. Embedded Representations

An embedded representation is encoded as an array that consists of the unsigned integer 0, followed by a byte string containing the representation data, optionally followed by representation metadata.

```
representation = [0, bytes, ?representation-metadata]
```

Representation metadata is encoded as an array of zero or more name-value pairs.

```
representation-metadata = [*(metadata-name, metadata-value)]
```

The metadata, if any, MUST be processed in a fresh environment. All variables in the new environment are initially set to a copy of the variables in the current environment.

The metadata name is defined in the same way as a link relation type (Section 3.1.2).

```
metadata-name = text
```

A metadata value can be a CoRI reference, a Boolean value, an integer, a floating-point number, a date/time value, a byte string, a

text string, or null. A CoRI reference MUST be resolved against the current base.

metadata-value = CoRI / literal

3.1.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. There is one type of directives available: the Base directive.

directive = base-directive

3.1.5.1. Base Directives

A Base directive is encoded as an array that consists of the unsigned integer 1, followed by a base.

base-directive = [1, base]

The base is denoted by a CoRI reference. This CoRI reference MUST be resolved against the current context (not the current base).

base = CoRI

The directive is processed by resolving the CoRI reference against the current context and assigning the result to the current base.

3.2. Dictionaries

The binary format can reference values from a dictionary to reduce representation size and processing cost. Dictionary references can be used in place of link relation types, link targets, operation types, submission targets, form field types, form field values, representation metadata names, and representation metadata values.

3.2.1. Dictionary References

A dictionary reference is encoded as an unsigned integer. Where a dictionary reference cannot be expressed unambiguously, the unsigned integer is tagged with CBOR tag TBD6.

relation-type /= uint

link-target /= #6.TBD6(uint)

operation-type /= uint

```

submission-target /= #6.TBD6(uint)

form-field-type /= uint

form-field-value /= #6.TBD6(uint)

metadata-name /= uint

metadata-value /= #6.TBD6(uint)

```

3.2.2. Media Type Parameter

The "application/coral+cbor" media type is defined to have a "dictionary" parameter that specifies the dictionary in use. The dictionary is identified by a URI [RFC3986]. For example, a CoRAL document that uses the dictionary identified by the URI <http://example.com/dictionary> can use the following content type:

```
application/coral+cbor;dictionary="http://example.com/dictionary"
```

The URI serves only as an identifier; it does not necessarily have to be dereferencable (or even use a dereferencable URI scheme). It is permissible, though, to use a dereferencable URI and to serve a representation that provides information about the dictionary in a human- or machine-readable way. (The format of such a representation is outside the scope of this document.)

For simplicity, a CoRAL document can reference values only from one dictionary; the value of the "dictionary" parameter MUST be a single URI. If the "dictionary" parameter is absent, the default dictionary specified in Appendix B of this document is assumed.

Once a dictionary has made an assignment, the assignment MUST NOT be changed or removed. A dictionary, however, may contain additional information about an assignment, which may change over time.

In CoAP [RFC7252], media types (including specific values for media type parameters) are encoded as an unsigned integer called "content format". For use with CoAP, each new CoRAL dictionary MUST register a new content format in the IANA CoAP Content-Formats Registry.

4. Textual Format

This section defines the syntax of documents in the CoRAL textual format using two grammars: The lexical grammar defines how Unicode characters are combined to form line terminators, white space, comments, and tokens. The syntactic grammar defines how tokens are

combined to form documents. Both grammars are presented in Augmented Backus-Naur Form (ABNF) [RFC5234].

A document in the textual format is a Unicode string in a Unicode encoding form [UNICODE]. The media type for such documents is "text/coral". The "charset" parameter is not used; charset information is transported inside the document in the form of an OPTIONAL Byte Order Mark (BOM). The use of the UTF-8 encoding scheme [RFC3629], without a BOM, is RECOMMENDED.

4.1. Lexical Structure

The lexical structure of a document in the textual format is made up of four basic elements: line terminators, white space, comments, and tokens. Of these, only tokens are significant in the syntactic grammar. There are five kinds of tokens: identifiers, IRIs, IRI references, literals, and punctuators.

token = identifier / iri / iriref / literal / punctuator

When several lexical grammar rules match a sequence of characters in a document, the longest match takes priority.

4.1.1. Line Terminators

Line terminators divide text into lines. A line terminator is any Unicode character with Line_Break class BK, CR, LF, or NL. However, any CR character that immediately precedes a LF character is ignored. (This affects only the numbering of lines in error messages.)

4.1.2. White Space

White space is a sequence of one or more white space characters. A white space character is any Unicode character with the White_Space property.

4.1.3. Comments

Comments are sequences of characters that are ignored when parsing text into tokens. Single-line comments begin with the characters "//" and extend to the end of the line. Delimited comments begin with the characters "/*" and end with the characters "*/". Delimited comments can occupy a portion of a line, a single line, or multiple lines.

Comments do not nest. The character sequences "/*" and "*/" have no special meaning within a single-line comment; the character sequences "//" and "/*" have no special meaning within a delimited comment.

4.1.4. Identifiers

An identifier token is a user-defined symbolic name. The rules for identifiers correspond to those recommended by the Unicode Standard Annex #31 [UNICODE-UAX31] using the following profile:

```

identifier = START *CONTINUE *(MEDIAL 1*CONTINUE)

START = <Any character with the XID_Start property>

CONTINUE = <Any character with the XID_Continue property>

MEDIAL = "-" / "." / "~" / %x58A / %xF0B

MEDIAL =/ %x2010 / %x2027 / %x30A0 / %x30FB

```

All identifiers MUST be converted into Unicode Normalization Form C (NFC), as defined by the Unicode Standard Annex #15 [UNICODE-UAX15]. Comparison of identifiers is based on NFC and is case-sensitive (unless otherwise noted).

4.1.5. IRIs and IRI References

IRIs and IRI references are Unicode strings that conform to the syntax defined in RFC 3987 [RFC3987]. An IRI reference can be either an IRI or a relative reference. Both IRIs and IRI references are enclosed in angle brackets ("**<**" and "**>**").

```

iri = "<" IRI ">"

iriref = "<" IRI-reference ">"

IRI = <Defined in Section 2.2 of RFC 3987>

IRI-reference = <Defined in Section 2.2 of RFC 3987>

```

4.1.6. Literals

A literal is a textual representation of a value. There are seven types of literals: Boolean, integer, floating-point, date/time, byte string, text string, and null.

```

literal = boolean / integer / float / datetime / bytes / text

literal =/ null

```

4.1.6.1. Boolean Literals

The case-insensitive tokens "true" and "false" denote the Boolean values true and false, respectively.

```
boolean = "true" / "false"
```

4.1.6.2. Integer Literals

Integer literals denote an integer value of unspecified precision. By default, integer literals are expressed in decimal, but they can also be specified in an alternate base using a prefix: Binary literals begin with "0b", octal literals begin with "0o", and hexadecimal literals begin with "0x".

Decimal literals contain the digits "0" through "9". Binary literals contain "0" and "1", octal literals contain "0" through "7", and hexadecimal literals contain "0" through "9" as well as "A" through "F" in upper- or lowercase.

Negative integers are expressed by prepending a minus sign ("-").

```
integer = ["+" / "-"] (decimal / binary / octal / hexadecimal)
```

```
decimal = 1 * DIGIT
```

```
binary = %x30 (%x42 / %x62) 1 * BINDIG
```

```
octal = %x30 (%x4F / %x6F) 1 * OCTDIG
```

```
hexadecimal = %x30 (%x58 / %x78) 1 * HEXDIG
```

```
DIGIT = %x30-39
```

```
BINDIG = %x30-31
```

```
OCTDIG = %x30-37
```

```
HEXDIG = %x30-39 / %x41-46 / %x61-66
```

4.1.6.3. Floating-point Literals

Floating-point literals denote a floating-point number of unspecified precision.

Floating-point literals consist of a sequence of decimal digits followed by a fraction, an exponent, or both. The fraction consists of a decimal point (".") followed by a sequence of decimal digits.

The exponent consists of the letter "e" in upper- or lowercase, followed by an optional sign and a sequence of decimal digits that indicate a power of 10 by which the value preceding the "e" is multiplied.

Negative floating-point values are expressed by prepending a minus sign ("-").

```
float = ["+" / "-"] 1*DIGIT [fraction] [exponent]
```

```
fraction = "." 1*DIGIT
```

```
exponent = (%x45 / %x65) ["+" / "-"] 1*DIGIT
```

A floating-point literal can additionally denote either the special "Not-a-Number" (NaN) value, positive infinity, or negative infinity. The NaN value is produced by the case-insensitive token "NaN". The two infinite values are produced by the case-insensitive tokens "+Infinity" (or simply "Infinity") and "-Infinity".

```
float =/ "NaN"
```

```
float =/ ["+" / "-"] "Infinity"
```

4.1.6.4. Date/Time Literals

Date/time literals denote an instant in time.

A date/time literal consists of the prefix "dt" and a sequence of Unicode characters in Internet Date/Time Format [RFC3339], enclosed in single quotes.

```
datetime = %x64.74 SQUOTE date-time SQUOTE
```

```
date-time = <Defined in Section 5.6 of RFC 3339>
```

```
SQUOTE = %x27
```

4.1.6.5. Byte String Literals

Byte string literals denote an ordered sequence of bytes.

A byte string literal consists of a prefix and zero or more bytes encoded in Base16, Base32, or Base64 [RFC4648], enclosed in single quotes. Byte string literals encoded in Base16 begin with "h" or "b16", byte string literals encoded in Base32 begin with "b32", and byte string literals encoded in Base64 begin with "b64".

bytes = base16 / base32 / base64

base16 = (%x68 / %x62.31.36) SQUOTE <Base16 encoded data> SQUOTE

base32 = %x62.33.32 SQUOTE <Base32 encoded data> SQUOTE

base64 = %x62.36.34 SQUOTE <Base64 encoded data> SQUOTE

4.1.6.6. Text String Literals

Text string literals denote a Unicode string.

A text string literal consists of zero or more Unicode characters enclosed in double quotes. It can include simple escape sequences (such as `\t` for the tab character) as well as hexadecimal and Unicode escape sequences.

text = DQUOTE *(char / %x5C escape) DQUOTE

char = <Any character except %x22, %x5C, and line terminators>

escape = simple-escape / hexadecimal-escape / unicode-escape

simple-escape = %x30 / %x62 / %x74 / %x6E / %x76

simple-escape = / %x66 / %x72 / %x22 / %x27 / %x5C

hexadecimal-escape = (%x78 / %x58) 2HEXDIG

unicode-escape = %x75 4HEXDIG / %x55 8HEXDIG

DQUOTE = %x22

An escape sequence denotes a single Unicode code point. For hexadecimal and Unicode escape sequences, the code point is expressed by the hexadecimal number following the `"\x"`, `"\X"`, `"\u"`, or `"\U"` prefix. Simple escape sequences indicate the code points listed in Table 1.

Escape Sequence	Code Point	Character Name
\0	U+0000	Null
\b	U+0008	Backspace
\t	U+0009	Character Tabulation
\n	U+000A	Line Feed
\v	U+000B	Line Tabulation
\f	U+000C	Form Feed
\r	U+000D	Carriage Return
\"	U+0022	Quotation Mark
\'	U+0027	Apostrophe
\\	U+005C	Reverse Solidus

Table 1: Simple Escape Sequences

4.1.6.7. Null Literal

The case-insensitive tokens "null" and "_" denote the intentional absence of any value.

```
null = "null" / "_"
```

4.1.7. Punctuators

Punctuator tokens are used for grouping and separating.

```
punctuator = "#" / ":" / "*" / "[" / "]" / "{" / "}" / "=" / "->"
```

4.2. Syntactic Structure

The syntactic structure of a document in the textual format is made up of four kinds of elements: links, forms, embedded representations, and (as an extension to the CoRAL data model) directives. Directives provide a way to make documents easier to read and write by setting a base for relative IRI references and introducing shorthands for IRIs.

Elements are processed in the order they appear in the document. Document processors need to maintain an `_environment_` while iterating a list of elements. The environment consists of three variables: the `_current context_`, the `_current base_`, and the `_current mapping from identifiers to IRIs_`. Both the current context and the current base are initially set to the document's retrieval context. The current mapping from identifiers to IRIs is initially empty.

4.2.1. Documents

The body of a document in the textual format consists of zero or more links, forms, embedded representations, and directives.

document = body

body = *(link / form / representation / directive)

4.2.2. Links

A link consists of the link relation type, followed by the link target, optionally followed by a link body enclosed in curly brackets ("{" and "}").

link = relation-type link-target [{" body "}"]

The link relation type is denoted by either an IRI, a simple name, or a qualified name.

relation-type = iri / simple-name / qualified-name

A simple name consists of an identifier. It is resolved to an IRI by looking up the empty string in the current mapping from identifiers to IRIs and appending the specified identifier to the result. It is an error if the empty string is not present in the current mapping.

simple-name = identifier

A qualified name consists of two identifiers separated by a colon (":"). It is resolved to an IRI by looking up the identifier on the left hand side in the current mapping from identifiers to IRIs and appending the identifier on the right hand side to the result. It is an error if the identifier on the left hand side is not present in the current mapping.

qualified-name = identifier ":" identifier

The link target is denoted by an IRI reference or represented by a value literal. An IRI reference MUST be resolved against the current base. If the link target is null, the link target is an unidentified resource.

link-target = iriref / literal

The list of elements in the link body, if any, MUST be processed in a fresh environment. Both the current context and current base in this environment are initially set to the link target of the enclosing

link. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

4.2.3. Forms

A form consists of the operation type, followed by a "->" token and the submission target, optionally followed by a list of form fields enclosed in square brackets "[" and "]").

form = operation-type "->" submission-target ["[" form-fields "]"]

The operation type is defined in the same way as a link relation type (Section 4.2.2).

operation-type = iri / simple-name / qualified-name

The request method is either implied by the operation type or encoded as a form field. If there are both, the form field takes precedence over the operation type. Either way, the method **MUST** be defined for the Web transfer protocol identified by the scheme of the submission target.

The submission target is denoted by an IRI reference. This IRI reference **MUST** be resolved against the current base.

submission-target = iriref

4.2.3.1. Form Fields

A list of form fields consists of zero or more type-value pairs.

form-fields = *(form-field-type form-field-value)

The list, if any, **MUST** be processed in a fresh environment. Both the current context and the current base in this environment are initially set to the submission target of the enclosing form. The mapping from identifiers to IRIs is initially set to a copy of the mapping from identifiers to IRIs in the current environment.

The form field type is defined in the same way as a link relation type (Section 4.2.2).

form-field-type = iri / simple-name / qualified-name

The form field value can be an IRI reference, Boolean literal, integer literal, floating-point literal, byte string literal, text

string literal, or null. An IRI reference MUST be resolved against the current base.

form-field-value = iriref / literal

4.2.4. Embedded Representations

An embedded representation consists of a "*" token, followed by the representation data, optionally followed by representation metadata enclosed in square brackets "[" and "]").

representation = "*" bytes [" representation-metadata "]"

Representation metadata consists of zero or more name-value pairs.

representation-metadata = *(metadata-name metadata-value)

The metadata, if any, MUST be processed in a fresh environment. All variables in the new environment are initially set to a copy of the variables in the current environment.

The metadata name is defined in the same way as a link relation type (Section 4.2.2).

metadata-name = iri / simple-name / qualified-name

The metadata value can be an IRI reference, Boolean literal, integer literal, floating-point literal, byte string literal, text string literal, or null. An IRI reference MUST be resolved against the current base.

metadata-value = iriref / literal

4.2.5. Directives

Directives provide the ability to manipulate the environment when processing a list of elements. All directives start with a number sign("#") followed by a directive identifier. Directive identifiers are case-insensitive and constrained to Unicode characters in the Basic Latin block.

The following two types of directives are available: the Base directive and the Using directive.

directive = base-directive / using-directive

4.2.5.1. Base Directives

A Base directive consists of a number sign ("#"), followed by the case-insensitive identifier "base", followed by a base.

```
base-directive = "#" "base" base
```

The base is denoted by an IRI reference. The IRI reference **MUST** be resolved against the current context (not the current base).

```
base = iriref
```

The directive is processed by resolving the IRI reference against the current context and assigning the result to the current base.

4.2.5.2. Using Directives

A Using directive consists of a number sign ("#"), followed by the case-insensitive identifier "using", optionally followed by an identifier and an equals sign ("="), finally followed by an IRI. If the identifier is not specified, it is assumed to be the empty string.

```
using-directive = "#" "using" [identifier "="] iri
```

The directive is processed by adding the specified identifier and IRI to the current mapping from identifiers to IRIs. It is an error if the identifier is already present in the mapping.

5. Usage Considerations

This section discusses some considerations in creating CoRAL-based applications and vocabularies.

5.1. Specifying CoRAL-based Applications

CoRAL-based applications naturally implement the Web architecture [W3C.REC-webarch-20041215] and thus are centered around orthogonal specifications for identification, interaction, and representation:

- o Resources are identified by IRIs or represented by value literals.
- o Interactions are based on the hypermedia interaction model of the Web and the methods provided by the Web transfer protocol. The semantics of possible interactions are identified by link relation types and operation types.

- o Representations are CoRAL documents encoded in the binary format defined in Section 3 or the textual format defined in Section 4. Depending on the application, additional representation formats may be used.

5.1.1. Application Interfaces

Specifications for CoRAL-based applications need to list the specific components used in the application interface and their identifiers. This should include the following items:

- o URI schemes that identify the Web transfer protocol(s) used in the application.
- o Internet media types that identify the representation format(s) used in the application, including the media type(s) of the CoRAL serialization format(s).
- o Link relation types that identify the semantics of links.
- o Operation types that identify the semantics of forms. Additionally, for each operation type, the permissible request method(s).
- o Form field types that identify the semantics of form fields. Additionally, for each form field type, the permissible form field values.
- o Metadata names that identify the semantics of representation metadata. Additionally, for each metadata name, the permissible metadata values.

5.1.2. Resource Identifiers

URIs [RFC3986] are a cornerstone of Web-based applications. They enable the uniform identification of resources and are used every time a client interacts with a server or a resource representation needs to refer to another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is a common practice in many HTTP-based application programming interfaces (APIs) to make this part of the application specification, i.e., to prescribe fixed URI templates that are hard-coded in implementations. There are a number of problems with this practice [RFC7320], though.

In CoRAL-based applications, resource names are therefore not part of the application specification -- they are an implementation detail. The specification of a CoRAL-based application MUST NOT mandate any particular form of resource name structure. BCP 190 [RFC7320] describes the problematic practice of fixed URI structures in more detail and provides some acceptable alternatives.

5.1.3. Implementation Limits

This document places no restrictions on the number of elements in a CoRAL document or the depth of nested elements. Applications using CoRAL (in particular those running in constrained environments) may wish to limit these numbers and specify implementation limits that an application implementation must at least support to be interoperable.

Applications may also mandate the following and other restrictions:

- o use of only either the binary format or the text format;
- o use of only either HTTP or CoAP as supported Web transfer protocol;
- o use of only dictionary references in the binary format for certain vocabulary;
- o use of only either content type strings or content format IDs;
- o use of CoRI references only up to a specific length;
- o use of CBOR in a canonical format (see Section 3.9 of RFC 7049).

5.2. Minting Vocabulary

New link relation types, operation types, form field types, and metadata names can be minted by defining an IRI [RFC3987] that uniquely identifies the item. Although the IRI can point to a resource that contains a definition of the semantics, clients SHOULD NOT automatically access that resource to avoid overburdening its server. The IRI SHOULD be under the control of the person or party defining it, or be delegated to them.

To avoid interoperability problems, it is RECOMMENDED that only IRIs are minted that are normalized according to Section 5.3 of RFC 3987. Non-normalized forms that are best avoided include:

- o Uppercase characters in scheme names and domain names

- o Percent-encoding of characters where it is not required by the IRI syntax
- o Explicitly stated HTTP default port (e.g., `<http://example.com/>` is preferable over `<http://example.com:80/>`)
- o Completely empty path in HTTP IRIs (e.g., `<http://example.com/>` is preferable over `<http://example.com>`)
- o Dot segments (`"./."` or `"/./"`) in the path component of an IRI
- o Lowercase hexadecimal letters within percent-encoding triplets (e.g., `"%3F"` is preferable over `"%3f"`)
- o Punycode-encoding of Internationalized Domain Names in IRIs
- o IRIs that are not in Unicode Normalization Form C [UNICODE-UAX15]

IRIs that identify vocabulary do not need to be registered. The inclusion of domain names in IRIs allows for the decentralized creation of new IRIs without the risk of collisions.

However, IRIs can be relatively verbose and impose a high overhead on a representation. This can be a problem in constrained environments [RFC7228]. Therefore, CoRAL alternatively allows the use of unsigned integers to reference CBOR data items from a dictionary, as specified in Section 3.2. These impose a much smaller overhead but instead need to be assigned by an authority to avoid collisions.

5.3. Expressing Registered Link Relation Types

Link relation types registered in the IANA Link Relations Registry, such as "collection" [RFC6573] or "icon" [W3C.REC-html52-20171214], can be used in CoRAL by appending the registered name to the IRI `<http://www.iana.org/assignments/relation/>`:

```
#using iana = <http://www.iana.org/assignments/relation/>

iana:collection </items>
iana:icon       </favicon.png>
```

Note that registered link relation types are required to be lowercased, as per Section 3.3 of RFC 8288 [RFC8288].

(The convention of appending the link relation types to the prefix `"http://www.iana.org/assignments/relation/"` to form IRIs is adopted from Atom [RFC4287]; see also Appendix A.2 of RFC 8288 [RFC8288].)

5.4. Expressing Simple RDF Statements

An RDF statement [W3C.REC-rdf11-concepts-20140225] says that some relationship, indicated by a predicate, holds between two resources. Existing RDF vocabularies can therefore be good source for link relation types that describe resource metadata. For example, a CoRAL document could use the FOAF vocabulary [FOAF] to describe the person or software that made it:

```
#using rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
#using foaf = <http://xmlns.com/foaf/0.1/>

foaf:maker null {
  rdf:type          <http://xmlns.com/foaf/0.1/Person>
  foaf:familyName   "Hartke"
  foaf:givenName    "Klaus"
  foaf:mbox         <mailto:klaus.hartke@ericsson.com>
}
```

5.5. Expressing Natural Language Texts

Text strings that are the target of a link can be associated with a language tag [RFC5646] and a base text direction (i.e., right-to-left or left-to-right) by nesting links of type `<http://coreapps.org/base#language>` and `<http://coreapps.org/base#direction>` under that link, respectively:

```
#using <http://coreapps.org/base#>
#using iana = <http://www.iana.org/assignments/relation/>

iana:terms-of-service </tos> {
  title "Nutzungsbedingungen" {
    language "de"
    direction "ltr"
  }
  title "Terms of use" {
    language "en-US"
    direction "ltr"
  }
}
```

The link relation types `<http://coreapps.org/base#language>` and `<http://coreapps.org/base#direction>` are defined in Appendix A.

5.6. Embedding CoRAL in CBOR Data

Data items in the CoRAL binary format (Section 3) may be embedded in other CBOR data [RFC7049] data. Specifications using CDDL [RFC8610] SHOULD reference the following CDDL definitions for this purpose:

CoRAL-Document = document

CoRAL-Link = link

CoRAL-Form = form

For each embedded document, link, and form, the retrieval context, link context, and form context needs to be specified, respectively.

5.7. Submitting CoRAL Documents

By default, a CoRAL document is a representation that captures the current state of a resource. The meaning of a CoRAL document changes when it is submitted in a request. Depending on the request method, the CoRAL document can capture the intended state of a resource (PUT) or be subject to application-specific processing (POST).

5.7.1. PUT Requests

A PUT request with a CoRAL document enclosed in the request payload requests that the state of the target resource be created or replaced with the state described by the CoRAL document. A successful PUT of a CoRAL document generally means that a subsequent GET on that same target resource would result in an equivalent document being sent in a success response.

An origin server SHOULD verify that a submitted CoRAL document is consistent with any constraints the server has for the target resource. When a document is inconsistent with the target resource, the origin server SHOULD either make it consistent (e.g., by removing inconsistent elements) or respond with an appropriate error message containing sufficient information to explain why the document is unsuitable.

The retrieval context and the base URI of a CoRAL document in a PUT are the request URI of the request.

5.7.2. POST Requests

A POST request with a CoRAL document enclosed in the request payload requests that the target resource process the CoRAL document according to the resource's own specific semantics.

The retrieval context of a CoRAL document in a POST is an unspecified URI. The base URI of the document is the request URI of the request.

5.8. Returning CoRAL Documents

In a response, the meaning of a CoRAL document changes depending on the request method and the response status code. For example, a CoRAL document in a successful response to a GET represents the current state of the target resource, whereas a CoRAL document in a successful response to a POST might represent either the processing result or the new resource state. A CoRAL document in an error response represents the error condition, usually describing the error state and what next steps are suggested for resolving it.

5.8.1. Success Responses

Success responses have a response status code that indicates that the client's request was successfully received, understood, and accepted. When the representation in a success response does not describe the state of the target resource, it describes result of processing the request.

For example, when a request has been fulfilled and has resulted in one or more new resources being created, a CoRAL document in the response can link to and describe the resource(s) created.

The retrieval context of a CoRAL document representing a processing result is an unspecified URI that refers to the processing result itself. The base URI of the document is the request URI of the request.

5.8.2. Error Responses

Error response have a response status code that indicates that either the request cannot be fulfilled or the server failed to fulfill an apparently valid request. A representation in an error response describes the error condition.

The retrieval context of such a CoRAL document representing an error condition is an unspecified URI that refers to the error condition itself. The base URI of the document is the request URI of the request.

6. Security Considerations

Parsers of CoRAL documents must operate on input that is assumed to be untrusted. This means that parsers MUST fail gracefully in the face of malicious inputs (e.g., inputs not adhering to the data

structure). Additionally, parsers MUST be prepared to deal with resource exhaustion (e.g., resulting from the allocation of big data items) or exhaustion of the call stack (stack overflow).

CoRAL serializations intentionally do not feature the equivalent of XML entity references as to preclude the whole class of attacks relating to these, such as exponential XML entity expansion ("billion laughs") [CAPEC-197] and malicious XML entity linking [CAPEC-201].

Implementers of the CoRAL binary format need to consider the security aspects of processing CBOR with the restrictions described in Section 3. Notably, different number representations for the same numeric value are not equivalent in the CoRAL binary format. See Section 8 of RFC 7049 [RFC7049] for security considerations relating to CBOR.

Implementers of the CoRAL textual format need to consider the security aspects of handling Unicode input. See the Unicode Standard Annex #36 [UNICODE-UAX36] for security considerations relating to visual spoofing and misuse of character encodings. See Section 10 of RFC 3629 [RFC3629] for security considerations relating to UTF-8.

CoRAL makes extensive use of resource identifiers. See Section 7 of RFC 3986 [RFC3986] for security considerations relating to URIs. See Section 8 of RFC 3987 [RFC3987] for security considerations relating to IRIs. See Section X of RFC XXXX [I-D.ietf-core-href] for security considerations relating to CoRIs.

The security of applications using CoRAL can depend on the proper preparation and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string. See RFC 6943 [RFC6943] for security considerations relating to identifiers in IRIs and other places.

CoRAL is intended to be used in conjunction with a Web transfer protocol like HTTP or CoAP. See Section 9 of RFC 7230 [RFC7230], Section 9 of RFC 7231 [RFC7231], etc., for security considerations relating to HTTP. See Section 11 of RFC 7252 [RFC7252] for security considerations relating to CoAP.

CoRAL does not define any specific mechanisms for protecting the confidentiality and integrity of CoRAL documents. It relies on application layer or transport layer mechanisms for this, such as Transport Layer Security (TLS) [RFC8446].

CoRAL documents and the structure of a web of resources revealed from automatically following links can disclose personal information and other sensitive information. Implementations need to prevent the unintentional disclosure of such information. See Section 9 of RFC 7231 [RFC7231] for additional considerations.

Applications using CoRAL ought to consider the attack vectors opened by automatically following, trusting, or otherwise using links and forms in CoRAL documents. Notably, a server that is authoritative for the CoRAL representation of a resource may not necessarily be authoritative for nested elements in the document. See Section 5 of RFC 8288 [RFC8288] for related considerations.

Unless an application mitigates this risk by specifying more specific rules, any link or form in a document where the link or form context and the document's retrieval context don't share the same Web origin [RFC6454] MUST be discarded ("same-origin policy").

7. IANA Considerations

7.1. Media Type "application/coral+cbor"

This document registers the media type "application/coral+cbor" according to the procedures of BCP 13 [RFC6838].

Type name:
application

Subtype name:
coral+cbor

Required parameters:
N/A

Optional parameters:
dictionary - See Section 3.2 of [I-D.ietf-core-coral].

Encoding considerations:
binary - See Section 3 of [I-D.ietf-core-coral].

Security considerations:
See Section 6 of [I-D.ietf-core-coral].

Interoperability considerations:
N/A

Published specification:
[I-D.ietf-core-coral]

Applications that use this media type:
See Section 1 of [I-D.ietf-core-coral].

Fragment identifier considerations:
As specified for "application/cbor".

Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): .coral.cbor
Macintosh file type code(s): N/A

Person & email address to contact for further information:
See the Author's Address section of [I-D.ietf-core-coral].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See the Author's Address section of [I-D.ietf-core-coral].

Change controller:
IESG

Provisional registration?
No

7.2. Media Type "text/coral"

This document registers the media type "text/coral" according to the procedures of BCP 13 [RFC6838] and guidelines in RFC 6657 [RFC6657].

Type name:
text

Subtype name:
coral

Required parameters:
N/A

Optional parameters:
N/A

Encoding considerations:

binary - See Section 4 of [I-D.ietf-core-coral].

Security considerations:

See Section 6 of [I-D.ietf-core-coral].

Interoperability considerations:

N/A

Published specification:

[I-D.ietf-core-coral]

Applications that use this media type:

See Section 1 of [I-D.ietf-core-coral].

Fragment identifier considerations:

N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .coral

Macintosh file type code(s): N/A

Person & email address to contact for further information:

See the Author's Address section of [I-D.ietf-core-coral].

Intended usage:

COMMON

Restrictions on usage:

N/A

Author:

See the Author's Address section of [I-D.ietf-core-coral].

Change controller:

IESG

Provisional registration?

No

7.3. CoAP Content Formats

This document registers CoAP content formats for the content types "application/coral+cbor" and "text/coral" according to the procedures of RFC 7252 [RFC7252].

- o Content Type: application/coral+cbor

Content Coding: identity
ID: TBD3
Reference: [I-D.ietf-core-coral]

- o Content Type: text/coral
Content Coding: identity
ID: TBD4
Reference: [I-D.ietf-core-coral]

[[NOTE TO RFC EDITOR: Please replace all occurrences of "TBD3" and "TBD4" in this document with the code points assigned by IANA.]]

[[NOTE TO IMPLEMENTERS: Experimental implementations can use content format ID 65087 for "application/coral+cbor" and content format ID 65343 for "text/coral" until IANA has assigned code points.]]

7.4. CBOR Tag

This document registers a CBOR tag for dictionary references according to the procedures of RFC 7049 [RFC7049].

- o Tag: TBD6
Data Item: unsigned integer
Semantics: Dictionary reference
Reference: [I-D.ietf-core-coral]

[[NOTE TO RFC EDITOR: Please replace all occurrences of "TBD6" in this document with the code point assigned by IANA.]]

8. References

8.1. Normative References

- [I-D.ietf-core-href]
Hartke, K., "Constrained Resource Identifiers", draft-ietf-core-href-01 (work in progress), November 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.

- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC6657] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, DOI 10.17487/RFC6657, July 2012, <<https://www.rfc-editor.org/info/rfc6657>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

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

[UNICODE] The Unicode Consortium, "The Unicode Standard", <<http://www.unicode.org/versions/latest/>>.

Note that this reference is to the latest version of Unicode, rather than to a specific release. It is not expected that future changes in the Unicode specification will have any impact on this document.

[UNICODE-UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", <<http://unicode.org/reports/tr15/>>.

[UNICODE-UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", <<http://unicode.org/reports/tr31/>>.

[UNICODE-UAX36] The Unicode Consortium, "Unicode Standard Annex #36: Unicode Security Considerations", <<http://unicode.org/reports/tr36/>>.

8.2. Informative References

[CAPEC-197] MITRE, "CAPEC-197: XML Entity Expansion", September 2019, <<https://capec.mitre.org/data/definitions/197.html>>.

[CAPEC-201] MITRE, "CAPEC-201: XML Entity Linking", September 2019, <<https://capec.mitre.org/data/definitions/201.html>>.

[FOAF] Brickley, D. and L. Miller, "FOAF Vocabulary Specification 0.99", January 2014, <<http://xmlns.com/foaf/spec/20140114.html>>.

[HAL] Kelly, M., "JSON Hypertext Application Language", draft-kelly-json-hal-08 (work in progress), May 2016.

- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <<https://www.rfc-editor.org/info/rfc4287>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<https://www.rfc-editor.org/info/rfc5789>>.
- [RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <<https://www.rfc-editor.org/info/rfc6573>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [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/info/rfc8132>>.

- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <<https://www.w3.org/TR/2017/REC-html52-20171214>>.
- [W3C.REC-rdf-schema-20140225]
Brickley, D. and R. Guha, "RDF Schema 1.1", World Wide Web Consortium Recommendation REC-rdf-schema-20140225, February 2014, <<http://www.w3.org/TR/2014/REC-rdf-schema-20140225>>.
- [W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium Recommendation REC-rdf11-concepts-20140225, February 2014, <<http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>>.
- [W3C.REC-turtle-20140225]
Prud'hommeaux, E. and G. Carothers, "RDF 1.1 Turtle", World Wide Web Consortium Recommendation REC-turtle-20140225, February 2014, <<http://www.w3.org/TR/2014/REC-turtle-20140225>>.
- [W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, December 2004, <<http://www.w3.org/TR/2004/REC-webarch-20041215>>.

Appendix A. Core Vocabulary

This section defines the core vocabulary for CoRAL: a set of link relation types, operation types, form field types, and metadata names.

A.1. Base

Link Relation Types:

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

Indicates that the link's context is an instance of the class specified as the link's target, as defined by RDF Schema [W3C.REC-rdf-schema-20140225].

<http://coreapps.org/base#title>

Indicates that the link target is a human-readable label (e.g., a menu entry).

The link target MUST be a text string. The text string SHOULD be annotated with a language and text direction using nested links of type <http://coreapps.org/base#language> and <http://coreapps.org/base#direction>, respectively.

<http://coreapps.org/base#language>

Indicates that the link target is a language tag [RFC5646] that specifies the language of the link context.

The link target MUST be a text string in the format specified in Section 2.1 of RFC 5646 [RFC5646].

<http://coreapps.org/base#direction>

Indicates that the link target is a base text direction (right-to-left or left-to-right) that specifies the text directionality of the link context.

The link target MUST be either the text string "rtl" or the text string "ltr".

Operation Types:

<http://coreapps.org/base#update>

Indicates that the state of the form's context can be replaced with the state described by a representation submitted to the server.

This operation type defaults to the PUT method [RFC7231] [RFC7252] for both HTTP and CoAP. Typical overrides by a form field include the PATCH method [RFC5789] [RFC8132] for HTTP and CoAP and the iPATCH method [RFC8132] for CoAP.

<http://coreapps.org/base#search>

Indicates that the form's context can be searched by submitting a search query.

This operation type defaults to the POST method [RFC7231] for HTTP and the FETCH method [RFC8132] for CoAP. Typical overrides by a form field include the POST method [RFC7252] for CoAP.

A.2. Collections

Link Relation Types:

<http://www.iana.org/assignments/relation/item>

Indicates that the link's context is a collection and that the link's target is a member of that collection, as defined in Section 2.1 of RFC 6573 [RFC6573].

<http://www.iana.org/assignments/relation/collection>

Indicates that the link's target is a collection and that the link's context is a member of that collection, as defined in Section 2.2 of RFC 6573 [RFC6573].

Operation Types:

<http://coreapps.org/collections#create>

Indicates that the form's context is a collection and that a new item can be created in that collection with the state defined by a representation submitted to the server.

This operation type defaults to the POST method [RFC7231] [RFC7252] for both HTTP and CoAP.

<http://coreapps.org/collections#delete>

Indicates that the form's context is a member of a collection and that the form's context can be removed from that collection.

This operation type defaults to the DELETE method [RFC7231] [RFC7252] for both HTTP and CoAP.

A.3. HTTP

Form Field Types:

<http://coreapps.org/http#method>

Specifies the HTTP method for the request.

The form field value MUST be a text string in the format defined in Section 4.1 of RFC 7231 [RFC7231]. The set of possible values is maintained in the IANA HTTP Method Registry.

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/http#accept>

Specifies an acceptable HTTP content type for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server accepts any or no request payload, depending on the operation type.

The form field value MUST be a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]. The possible set of media types and their parameters are maintained in the IANA Media Types Registry.

Representation Metadata:

<http://coreapps.org/http#type>

Specifies the HTTP content type of the representation.

The metadata value MUST be specified as a text string in the format defined in Section 3.1.1.1 of RFC 7231 [RFC7231]. The possible set of media types and their parameters are maintained in the IANA Media Types Registry.

Metadata of this type MUST NOT occur more than once for a representation. If absent, its value defaults to content type "application/octet-stream".

A.4. CoAP

Form Field Types:

<http://coreapps.org/coap#method>

Specifies the CoAP method for the request.

The form field value MUST be an integer identifying one of the CoAP request methods maintained in the IANA CoAP Method Codes Registry (e.g., the integer 2 for the POST method).

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/coap#accept>

Specifies an acceptable CoAP content format for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server

accepts any or no request payload, depending on the operation type.

The form field value MUST be an integer identifying one of the content formats maintained in the IANA CoAP Content-Formats Registry.

Representation Metadata:

<http://coreapps.org/coap#type>

Specifies the CoAP content format of the representation.

The metadata value MUST be an integer identifying one of the content formats maintained in the IANA CoAP Content-Formats Registry.

Metadata of this type MUST NOT occur more than once for a representation. If absent, it defaults to content format 42 (i.e., content type "application/octet-stream" without a content coding).

Appendix B. Default Dictionary

This section defines a default dictionary that is assumed when the "application/coral+cbor" media type is used without a "dictionary" parameter.

Key	Value
0	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
1	<http://www.iana.org/assignments/relation/item>
2	<http://www.iana.org/assignments/relation/collection>
3	<http://coreapps.org/collections#create>
4	<http://coreapps.org/base#update>
5	<http://coreapps.org/collections#delete>
6	<http://coreapps.org/base#search>
7	<http://coreapps.org/coap#accept>
8	<http://coreapps.org/coap#type>
9	<http://coreapps.org/base#language>
10	<http://coreapps.org/coap#method>
11	<http://coreapps.org/base#direction>
12	"ltr"
13	"rtl"

Table 2: Default Dictionary

Acknowledgements

CoRAL is heavily inspired by Mike Kelly's JSON Hypertext Application Language [HAL].

This document has benefited greatly from discussions and reviews of the CoRAL design team:

Christian Amsuess
<christian@amsuess.com>

Carsten Bormann, Universitaet Bremen
<cabo@tzi.org>

Michael Koster, SmartThings
<michael.koster@smarththings.com>

Jim Schaad, August Cellars
<ietf@augustcellars.com>

Thanks to Thomas Fossati, Jaime Jimenez, Sebastian Kaebisch, Ari Keranen, Matthias Kovatsch, and Niklas Widell for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: May 7, 2020

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB
November 04, 2019

CoAP: Echo, Request-Tag, and Token Processing
draft-ietf-core-echo-request-tag-08

Abstract

This document specifies enhancements to the Constrained Application Protocol (CoAP) that mitigate security issues in particular use cases. The Echo option enables a CoAP server to verify the freshness of a request or to force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match block-wise message fragments belonging to the same request. The update to the client Token processing requirements of RFC 7252 forbids non-secure reuse of Tokens to ensure binding of responses to requests when CoAP is used with security.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
2. Request Freshness and the Echo Option	4
2.1. Request Freshness	4
2.2. The Echo Option	5
2.2.1. Echo Option Format	5
2.3. Echo Processing	6
2.4. Applications of the Echo Option	10
3. Protecting Message Bodies using Request Tags	11
3.1. Fragmented Message Body Integrity	11
3.2. The Request-Tag Option	12
3.2.1. Request-Tag Option Format	12
3.3. Request-Tag Processing by Servers	13
3.4. Setting the Request-Tag	14
3.5. Applications of the Request-Tag Option	15
3.5.1. Body Integrity Based on Payload Integrity	15
3.5.2. Multiple Concurrent Block-wise Operations	16
3.5.3. Simplified Block-Wise Handling for Constrained Proxies	16
3.6. Rationale for the Option Properties	16
3.7. Rationale for Introducing the Option	17
3.8. Block2 / ETag Processing	17
4. Token Processing for Secure Request-Response Binding	18
4.1. Request-Response Binding	18
4.2. Updated Token Processing Requirements for Clients	18
5. Security Considerations	19
5.1. Token reuse	20
6. Privacy Considerations	21
7. IANA Considerations	21
8. References	22
8.1. Normative References	22
8.2. Informative References	22
Appendix A. Methods for Generating Echo Option Values	23
Appendix B. Request-Tag Message Size Impact	25
Appendix C. Change Log	25
Acknowledgments	29
Authors' Addresses	29

1. Introduction

The initial Constrained Application Protocol (CoAP) suite of specifications ([RFC7252], [RFC7641], and [RFC7959]) was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS ([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations. This document specifies security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two CoAP options, the Echo option and the Request-Tag option: The Echo option enables a CoAP server to verify the freshness of a request, synchronize state, or force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request, fragmented using the CoAP block-wise Transfer mechanism, which mitigates attacks and enables concurrent block-wise operations. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected using e.g. DTLS ([RFC6347]), TLS ([RFC8446]), or OSCORE ([RFC8613]), provide the additional security features.

The document also updates the Token processing requirements for clients specified in [RFC7252]. The updated processing forbids non-secure reuse of Tokens to ensure binding of responses to requests when CoAP is used with security, thus mitigating error cases and attacks where the client may erroneously associate the wrong response to a request.

Each of the following sections provides a more detailed introduction to the topic at hand in its first subsection.

1.1. Terminology

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.

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252]. The term "origin server" is used as in [RFC7252]. The term "origin client" is used in this document to denote the client from which a request originates; to distinguish from clients in proxies.

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation". An operation fragmented using [RFC7959] is called a "block-wise operation". A block-wise operation which is fragmenting the request body is called a "block-wise request operation". A block-wise operation which is fragmenting the response body is called a "block-wise response operation".

Two request messages are said to be "matchable" if they occur between the same endpoint pair, have the same code and the same set of options except for elective NoCacheKey options and options involved in block-wise transfer (Block1, Block2 and Request-Tag). Two operations are said to be matchable if any of their messages are.

Two matchable block-wise operations are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent block-wise request operations from a single endpoint are impossible with the options of [RFC7959] (see the last paragraphs of Sections 2.4 and 2.5) because the second operation's block overwrites any state of the first exchange.).

The Echo and Request-Tag options are defined in this document.

2. Request Freshness and the Echo Option

2.1. Request Freshness

A CoAP server receiving a request is in general not able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly perilous in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks can be mitigated by establishing fresh session keys, e.g. performing a DTLS handshake for each request, but in general this is not a solution suitable for constrained environments, for example, due to increased message overhead and latency. Additionally, if there are proxies, fresh DTLS session keys between server and proxy does not say anything about when the client made the request. In a general hop-by-hop setting, freshness may need to be verified in each hop.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time.

2.2. The Echo Option

This document defines the Echo option, a a lightweight challenge-response mechanism for CoAP that enables a CoAP server to verify the freshness of a request. A fresh request is one whose age has not yet exceeded the freshness requirements set by the server. The freshness requirements are application specific and may vary based on resource, method, and parameters outside of CoAP such as policies. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed back to the server in one or more CoAP requests. The Echo option provides a convention to transfer freshness indicators that works for all CoAP methods and response codes.

This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing state between CoAP client and server, cryptographically verify the aliveness of the client, or force a client to demonstrate reachability at its claimed network address. The same functionality can be provided by echoing freshness indicators in CoAP payloads, but this only works for methods and response codes defined to have a payload. The Echo option provides a convention to transfer freshness indicators that works for all methods and response codes.

2.2.1. Echo Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1, which extends Table 4 of [RFC7252]).

No.	C	U	N	R	Name	Format	Len.	Default	E	U
TBD			x		Echo	opaque	4-40	(none)	x	x

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 1: Echo Option Summary

The Echo option value is generated by a server, and its content and structure are implementation specific. Different methods for generating Echo option values are outlined in Appendix A. Clients and intermediaries MUST treat an Echo option value as opaque and make no assumptions about its content or structure.

When receiving an Echo option in a request, the server MUST be able to verify that the Echo option value (a) was generated by the server or some other party that the server trusts, and (b) fulfills the freshness requirements of the application. Depending on the freshness requirements the server may verify exactly when the Echo option value was generated (time-based freshness) or verify that the Echo option was generated after a specific event (event-based freshness). As the request is bound to the Echo option value, the server can determine that the request is not older than the Echo option value.

When the Echo option is used with OSCORE [RFC8613] it MAY be an Inner or Outer option, and the Inner and Outer values are independent. OSCORE servers MUST only produce Inner Echo options unless they are merely testing for reachability of the client (the same as proxies may do). The Inner option is encrypted and integrity protected between the endpoints, whereas the Outer option is not protected by OSCORE and visible between the endpoints to the extent it is not protected by some other security protocol. E.g. in the case of DTLS hop-by-hop between the endpoints, the Outer option is visible to proxies along the path.

2.3. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.4 for different applications).

The application decides under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

If a certain request is required to be fresh, the request does not contain a fresh Echo option value, and the server cannot verify the freshness of the request in some other way, the server MUST NOT process the request further and SHOULD send a 4.01 Unauthorized response with an Echo option. The server MAY include the same Echo option value in several different response messages and to different clients. Examples of this could be time-based freshness when several responses are sent closely after each other or event-based freshness with no event taking place between the responses.

The server may use request freshness provided by the Echo option to verify the aliveness of a client or to synchronize state. The server may also include the Echo option in a response to force a client to demonstrate reachability at its claimed network address. Note that the Echo option does not bind a request to any particular previous

response, but provides an indication that the client had access to the previous response at the time when it created the request.

Upon receiving a 4.01 Unauthorized response with the Echo option, the client SHOULD resend the original request with the addition of an Echo option with the received Echo option value. The client MAY send a different request compared to the original request. Upon receiving any other response with the Echo option, the client SHOULD echo the Echo option value in the next request to the server. The client MAY include the same Echo option value in several different requests to the server.

A client MUST only send Echo values to endpoints it received them from (where as defined in [RFC7252] Section 1.2, the security association is part of the endpoint). In OSCORE processing, that means sending Echo values from Outer options (or from non-OSCORE responses) back in Outer options, and those from Inner options in Inner options in the same security context.

Upon receiving a request with the Echo option, the server determines if the request is required to be fresh. If not, the Echo option MAY be ignored. If the request is required to be fresh and the server cannot verify the freshness of the request in some other way, the server MUST use the Echo option to verify that the request is fresh. If the server cannot verify that the request is fresh, the request is not processed further, and an error message MAY be sent. The error message SHOULD include a new Echo option.

One way for the server to verify freshness is that to bind the Echo value to a specific point in time and verify that the request is not older than a certain threshold T . The server can verify this by checking that $(t1 - t0) < T$, where $t1$ is the request receive time and $t0$ is the time when the Echo option value was generated. An example message flow is shown in Figure 2.

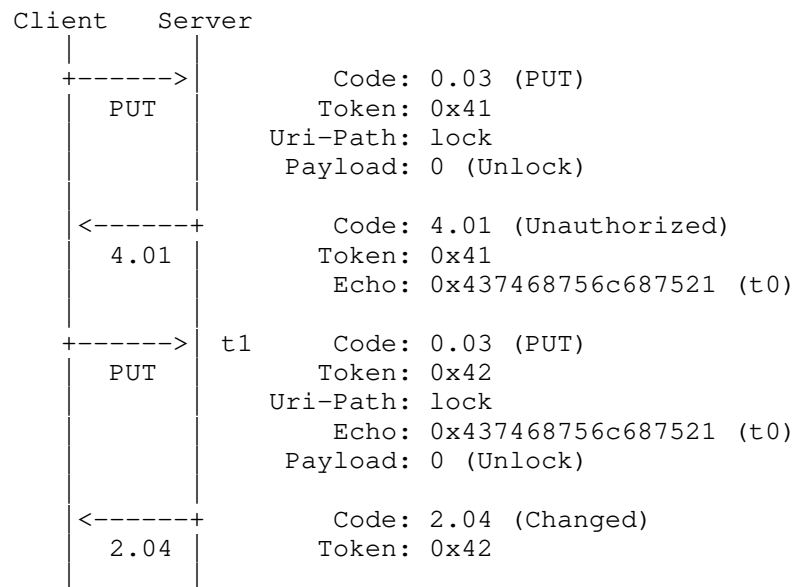


Figure 2: Example Message Flow for Time-Based Freshness

Another way for the server to verify freshness is to maintain a cache of values associated to events. The size of the cache is defined by the application. In the following we assume the cache size is 1, in which case freshness is defined as no new event has taken place. At each event a new value is written into the cache. The cache values MUST be different for all practical purposes. The server verifies freshness by checking that e_0 equals e_1 , where e_0 is the cached value when the Echo option value was generated, and e_1 is the cached value at the reception of the request. An example message flow is shown in Figure 3.

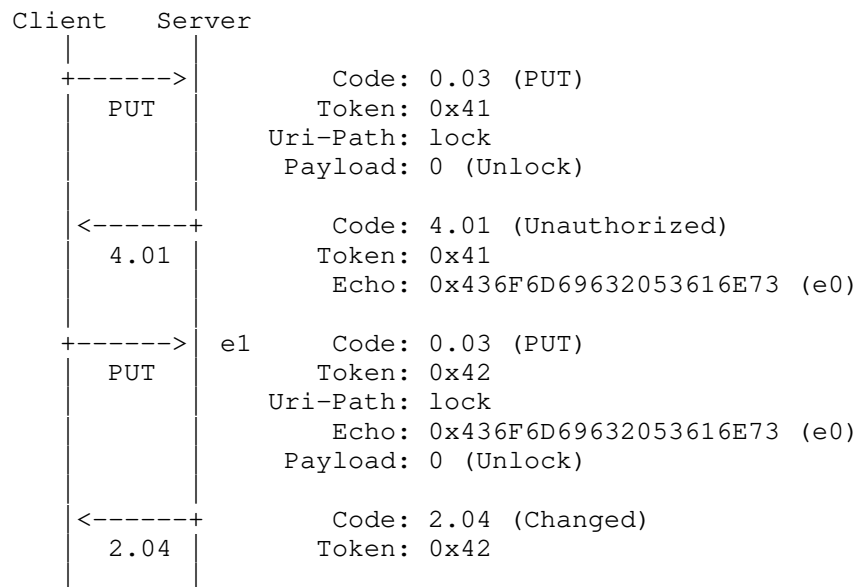


Figure 3: Example Message Flow for Event-Based Freshness

When used to serve freshness requirements (including client aliveness and state synchronizing), the Echo option value **MUST** be integrity protected between the intended endpoints, e.g. using DTLS, TLS, or an OSCORE Inner option ([RFC8613]). When used to demonstrate reachability at a claimed network address, the Echo option **SHOULD** contain the client's network address, but **MAY** be unprotected.

A CoAP-to-CoAP proxy **MAY** set an Echo option on responses, both on forwarded ones that had no Echo option or ones generated by the proxy (from cache or as an error). If it does so, it **MUST** remove the Echo option it recognizes as one generated by itself on follow-up requests. However, it **MUST** relay the Echo option of responses unmodified, and **MUST** relay the Echo option of requests it does not recognize as generated by itself unmodified.

The CoAP server side of CoAP-to-HTTP proxies **MAY** request freshness, especially if they have reason to assume that access may require it (e.g. because it is a PUT or POST); how this is determined is out of scope for this document. The CoAP client side of HTTP-to-CoAP proxies **SHOULD** respond to Echo challenges themselves if they know from the recent establishing of the connection that the HTTP request is fresh. Otherwise, they **SHOULD** respond with 503 Service Unavailable, Retry-After: 0 and terminate any underlying Keep-Alive connection. They **MAY** also use other mechanisms to establish freshness of the HTTP request that are not specified here.

2.4. Applications of the Echo Option

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions. In general, all non-safe methods (e.g. POST, PUT, DELETE) may require freshness guarantees for secure operation.
 - * The same Echo value may be used for multiple actuation requests to the same server, as long as the total round-trip time since the Echo option value was generated is below the freshness threshold.
 - * For actuator applications with low delay tolerance, to avoid additional round-trips for multiple requests in rapid sequence, the server may include the Echo option with a new value even in a successful response to a request, irrespectively of whether the request contained an Echo option or not. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.
2. A server may use the Echo option to synchronize properties (such as state or time) with a requesting client. A server **MUST NOT** synchronize a property with a client which is not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then server **MUST NOT** synchronize time with a client requesting access unless it is time authority for the server.
 - * If a server reboots during operation it may need to synchronize state or time before continuing the interaction. For example, with OSCORE it is possible to reuse a partly persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option, see Section 7.5 of [RFC8613].
 - * A device joining a CoAP group communication [RFC7390] protected with OSCORE [I-D.ietf-core-oscore-groupcomm] may be required to initially verify freshness and synchronize state or time with a client by using the Echo option in a unicast response to a multicast request. The client receiving the response with the Echo option includes the Echo option with the same value in a request, either in a unicast request to the responding server, or in a subsequent group request. In the latter case, the Echo option will be ignored except by the responding server.

3. A server that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks such as described in Section 11.3 of [RFC7252] (where an attacker would put a victim's address in the source address of a CoAP request). For this purpose, a server MAY ask a client to Echo its request to verify its source address. This needs to be done only once per peer and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Echo option can be used in messages that are not integrity protected, for example during discovery.

- * In the presence of a proxy, a server will not be able to distinguish different origin client endpoints. Following from the recommendation above, a proxy that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks. The proxy MAY use Echo to verify origin reachability as described in Section 2.3. The proxy MAY forward idempotent requests immediately to have a cached result available when the client's Echoed request arrives.

4. A server may want to use the request freshness provided by the Echo to verify the aliveness of a client. Note that in a deployment with hop-by-hop security and proxies, the server can only verify aliveness of the closest proxy.

3. Protecting Message Bodies using Request Tags

3.1. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The block-wise transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request and response payload, respectively. The block-wise functionality does not support the detection of interchanged blocks between different message bodies to the same resource having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCORE, within the replay window

([I-D.mattsson-core-coap-actuators]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option.

3.2. The Request-Tag Option

This document defines the Request-Tag option for identifying request bodies, similar to ETag, but ephemeral and set by the CoAP client. The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct block-wise request operations on one resource from one client, addressing the issue described in Section 3.1. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous block-wise request operations on a single resource. The requests must be integrity protected if they should protect against interchange of blocks between different message bodies. The Request-Tag option is only used in requests that carry the Block1 option, and in Block2 requests following these.

In essence, it is an implementation of the "proxy-safe elective option" used just to "vary the cache key" as suggested in [RFC7959] Section 2.4.

3.2.1. Request-Tag Option Format

The Request-Tag option is not critical, is safe to forward, repeatable, and part of the cache key, see Figure 4, which extends Table 4 of [RFC7252]).

No.	C	U	N	R	Name	Format	Len.	Default	E	U
TBD				x	Request-Tag	opaque	0-8	(none)	x	x

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 4: Request-Tag Option Summary

Request-Tag, like the block options, is both a class E and a class U option in terms of OSCORE processing (see Section 4.1 of [RFC8613]):

The Request-Tag MAY be an Inner or Outer option. It influences the Inner or Outer block operation, respectively. The Inner and Outer values are therefore independent of each other. The Inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The Outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

The Request-Tag option is only used in the request messages of block-wise operations.

The Request-Tag mechanism can be applied independently on the server and client sides of CoAP-to-CoAP proxies as are the block options, though given it is safe to forward, a proxy is free to just forward it when processing an operation. CoAP-to-HTTP proxies and HTTP-to-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

3.3. Request-Tag Processing by Servers

The Request-Tag option does not require any particular processing on the server side outside of the processing already necessary for any unknown elective proxy-safe cache-key option: The option varies the properties that distinguish block-wise operations (which includes all options except elective NoCacheKey and except Block1/2), and thus the server can not treat messages with a different list of Request-Tag options as belonging to the same operation.

To keep utilizing the cache, a server (including proxies) MAY discard the Request-Tag option from an assembled block-wise request when consulting its cache, as the option relates to the operation-on-the-wire and not its semantics. For example, a FETCH request with the same body as an older one can be served from the cache if the older's Max-Age has not expired yet, even if the second operation uses a Request-Tag and the first did not. (This is similar to the situation about ETag in that it is formally part of the cache key, but implementations that are aware of its meaning can cache more efficiently, see [RFC7252] Section 5.4.2).

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages carrying the same Request-Tag is a necessary but not sufficient condition for being part of the same operation. For one, a server may still treat them as independent messages when it sends 2.01/2.04 responses for every block. Also, a client that lost interest in an old operation but wants to start over can overwrite

the server's old state with a new initial (num=0) Block1 request and the same Request-Tag under some circumstances. Likewise, that results in the new message not being part of the old operation.

As it has always been, a server that can only serve a limited number of block-wise operations at the same time can delay the start of the operation by replying with 5.03 (Service unavailable) and a Max-Age indicating how long it expects the existing operation to go on, or it can forget about the state established with the older operation and respond with 4.08 (Request Entity Incomplete) to later blocks on the first operation.

3.4. Setting the Request-Tag

For each separate block-wise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. It needs to be set to the same value (or unset) in all messages belonging to the same operation, as otherwise they are treated as separate operations by the server.

Starting a request operation matchable to a previous operation and even using the same Request-Tag value is called request tag recycling. The absence of a Request-Tag option is viewed as a value distinct from all values with a single Request-Tag option set; starting a request operation matchable to a previous operation where neither has a Request-Tag option therefore constitutes request tag recycling just as well (also called "recycling the absent option").

Clients that use Request-Tag for a particular purpose (like in Section 3.5) MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on that purpose, and is defined there.

When Block1 and Block2 are combined in an operation, the Request-Tag of the Block1 phase is set in the Block2 phase as well for otherwise the request would have a different set of options and would not be recognized any more.

Clients are encouraged to generate compact messages. This means sending messages without Request-Tag options whenever possible, and using short values when the absent option can not be recycled.

The Request-Tag options MAY be present in request messages that carry a Block2 option even if those messages are not part of a blockwise request operation (this is to allow the operation described in Section 3.5.3). The Request-Tag option MUST NOT be present in response messages, and MUST NOT be present if neither the Block1 nor the Block2 option is present.

3.5. Applications of the Request-Tag Option

3.5.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace a later operation's blocks with an earlier operation's blocks (see Section 2.5 of [I-D.mattsson-core-coap-actuators]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- o The individual exchanges MUST be integrity protected end-to-end between client and server.
- o The client MUST NOT recycle a request tag in a new operation unless the previous operation matchable to the new one has concluded.

If any future security mechanisms allow a block-wise transfer to continue after an endpoint's details (like the IP address) have changed, then the client MUST consider messages sent to `_any_` endpoint address within the new operation's security context.

- o The client MUST NOT regard a block-wise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

Typically, in OSCORE, these confirmations can result either from the client receiving an OSCORE response message matching the request (an empty ACK is insufficient), or because the message's sequence number is old enough to be outside the server's receive window.

In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

Authors of other documents (e.g. applications of [RFC8613]) are invited to mandate this behavior for clients that execute block-wise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [RFC8323]). This is because with each message, any earlier message can not be replayed any more, so the client never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.5.2. Multiple Concurrent Block-wise Operations

CoAP clients, especially CoAP proxies, may initiate a block-wise request operation to a resource, to which a previous one is already in progress, which the new request should not cancel. A CoAP proxy would be in such a situation when it forwards operations with the same cache-key options but possibly different payloads.

For those cases, Request-Tag is the proxy-safe elective option suggested in [RFC7959] Section 2.4 last paragraph.

When initializing a new block-wise operation, a client has to look at other active operations:

- o If any of them is matchable to the new one, and the client neither wants to cancel the old one nor postpone the new one, it can pick a Request-Tag value (including the absent option) that is not in use by the other matchable operations for the new operation.
- o Otherwise, it can start the new operation without setting the Request-Tag option on it.

3.5.3. Simplified Block-Wise Handling for Constrained Proxies

The Block options were defined to be unsafe to forward because a proxy that would forward blocks as plain messages would risk mixing up clients' requests.

The Request-Tag option provides a very simple way for a proxy to keep them separate: if it appends a Request-Tag that is particular to the requesting endpoint to all request carrying any Block option, it does not need to keep track of any further block state.

This is particularly useful to proxies that strive for stateless operation as described in [I-D.ietf-core-stateless] Section 3.1.

3.6. Rationale for the Option Properties

The Request-Tag option can be elective, because to servers unaware of the Request-Tag option, operations with differing request tags will not be matchable.

The Request-Tag option can be safe to forward but part of the cache key, because to proxies unaware of the Request-Tag option will consider operations with differing request tags unmatchable but can still forward them.

The Request-Tag option is repeatable because this easily allows stateless proxies to "chain" their origin address. They can perform the steps of Section 3.5.3 without the need to create an option value that is the concatenation of the received option and their own value, and can simply add a new Request-Tag option unconditionally.

In draft versions of this document, the Request-Tag option used to be critical and unsafe to forward. That design was based on an erroneous understanding of which blocks could be composed according to [RFC7959].

3.7. Rationale for Introducing the Option

An alternative that was considered to the Request-Tag option for coping with the problem of fragmented message body integrity (Section 3.5.1) was to update [RFC7959] to say that blocks could only be assembled if their fragments' order corresponded to the sequence numbers.

That approach would have been difficult to roll out reliably on DTLS where many implementations do not expose sequence numbers, and would still not prevent attacks like in [I-D.mattsson-core-coap-actuators] Section 2.5.2.

3.8. Block2 / ETag Processing

The same security properties as in Section 3.5.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Rules stating that response body reassembly is conditional on matching ETag values are already in place from Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.5.1, a server MUST use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and MUST NOT use the same ETag value for different representations of a resource.

4. Token Processing for Secure Request-Response Binding

4.1. Request-Response Binding

A fundamental requirement of secure REST operations is that the client can bind a response to a particular request. If this is not ensured, a client may erroneously associate the wrong response to a request. The wrong response may be an old response for the same resource or for a completely different resource (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). For example, a request for the alarm status "GET /status" may be associated to a prior response "on", instead of the correct response "off".

In HTTPS, this type of binding is always assured by the ordered and reliable delivery as well as mandating that the server sends responses in the same order that the requests were received. The same is not true for CoAP where the server (or an attacker) can return responses in any order and where there can be any number of responses to a request (see e.g. [RFC7641]). In CoAP, concurrent requests are differentiated by their Token. Note that the CoAP Message ID cannot be used for this purpose since those are typically different for REST request and corresponding response in case of "separate response", see Section 2.2 of [RFC7252].

CoAP [RFC7252] does not treat Token as a cryptographically important value and does not give stricter guidelines than that the Tokens currently "in use" SHOULD (not SHALL) be unique. If used with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) Token reuse may result in situations where a client matches a response to the wrong request. Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

A straightforward mitigation is to mandate clients to not reuse Tokens until the traffic keys have been replaced. One easy way to accomplish this is to implement the Token as a counter starting at zero for each new or rekeyed secure connection.

4.2. Updated Token Processing Requirements for Clients

As described in Section 4.1, the client must be able to verify that a response corresponds to a particular request. This section updates the Token processing requirements for clients in [RFC7252] to always assure a cryptographically secure binding of responses to requests for secure REST operations like "coaps". The Token processing for servers is not updated. Token processing in Section 5.3.1 of [RFC7252] is updated by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the Tokens have cryptographic importance. The client MUST make sure that Tokens are not used in a way so that responses risk being associated with the wrong request. One easy way to accomplish this is to implement the Token (or part of the Token) as a sequence number starting at zero for each new or rekeyed secure connection, this approach SHOULD be followed.

5. Security Considerations

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of the Echo option. If no true random number generator is available, a truly random seed must be provided from an external source. As each pseudorandom number must only be used once, an implementation need to get a new truly random seed after reboot, or continuously store state in nonvolatile memory, see ([RFC8613], Appendix B.1.1) for issues and solution approaches for writing to nonvolatile memory.

A single active Echo value with 64 (pseudo-)random bits gives the same theoretical security level as a 64-bit MAC (as used in e.g. AES_128_CCM_8). The Echo option value MUST contain 32 (pseudo-)random bits that are not predictable for any other party than the server, and SHOULD contain 64 (pseudo-)random bits. A server MAY use different security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

The security provided by the Echo and Request-Tag options depends on the security protocol used. CoAP and HTTP proxies require (D)TLS to be terminated at the proxies. The proxies are therefore able to manipulate, inject, delete, or reorder options or packets. The security claims in such architectures only hold under the assumption that all intermediaries are fully trusted and have not been compromised.

Servers SHOULD use a monotonic clock to generate timestamps and compute round-trip times. Use of non-monotonic clocks is not secure as the server will accept expired Echo option values if the clock is moved backward. The server will also reject fresh Echo option values if the clock is moved forward. Non-monotonic clocks MAY be used as long as they have deviations that are acceptable given the freshness requirements. If the deviations from a monotonic clock are known, it may be possible to adjust the threshold accordingly.

An attacker may be able to affect the server's system time in various ways such as setting up a fake NTP server or broadcasting false time signals to radio-controlled clocks.

Servers MAY use the time since reboot measured in some unit of time. Servers MAY reset the timer at certain times and MAY generate a random offset applied to all timestamps. When resetting the timer, the server MUST reject all Echo values that was created before the reset.

Servers that use the List of Cached Random Values and Timestamps method described in Appendix A may be vulnerable to resource exhaustion attacks. One way to minimize state is to use the Integrity Protected Timestamp method described in Appendix A.

5.1. Token reuse

Reusing Tokens in a way so that responses are guaranteed to not be associated with the wrong request is not trivial as on-path attackers may block, delay, and reorder messages, requests may be sent to several servers, and servers may process requests in any order and send many responses to the same request. The use of a sequence number is therefore recommended when CoAP is used with a security protocol that does not providing bindings between requests and responses such as DTLS or TLS.

For a generic response to a confirmable request over DTLS, binding can only be claimed without out-of-band knowledge if

- o the original request was never retransmitted,
- o the response was piggybacked in an Acknowledgement message (as a confirmable or non-confirmable response may have been transmitted multiple times), and
- o if observation was used, the same holds for the registration, all re-registrations, and the cancellation.

(In addition, for observations, any responses using that Token and a DTLS sequence number earlier than the cancellation Acknowledgement message need to be discarded. This is typically not supported in DTLS implementations.)

In some setups, Tokens can be reused without the above constraints, as a different component in the setup provides the associations:

- o In CoAP over TLS, retransmissions are not handled by the CoAP layer and the replay window size is always exactly 1. When a client is sending TLS protected requests without Observe to a single server, the client can reuse a Token as soon as the previous response with that Token has been received.

- o Requests whose responses are cryptographically bound to the requests (like in OSCORE) can reuse Tokens indefinitely.

In all other cases, a sequence number approach is RECOMMENDED as per Section 4.

Tokens that cannot be reused need to be handled appropriately. This could be solved by increasing the Token as soon as the currently used Token cannot be reused, or by keeping a list of all blacklisted Tokens.

When the Token (or part of the Token) contains a sequence number, the encoding of the sequence number has to be chosen in a way to avoid any collisions. This is especially true when the Token contains more information than just the sequence number, e.g. serialized state as in [I-D.ietf-core-stateless].

6. Privacy Considerations

Implementations SHOULD NOT put any privacy sensitive information in the Echo or Request-Tag option values. Unencrypted timestamps MAY reveal information about the server such as location or time since reboot, or that the server will accept expired certificates. Timestamps MAY be used if Echo is encrypted between the client and the server, e.g. in the case of DTLS without proxies or when using OSCORE with an Inner Echo option.

Like HTTP cookies, the Echo option could potentially be abused as a tracking mechanism to link to different requests to the same client. This is especially true for pre-emptive Echo values. Servers MUST NOT use the Echo option to correlate requests for other purposes than freshness and reachability. Clients only send Echo to the same from which they were received. Compared to HTTP, CoAP clients are often authenticated and non-mobile, and servers can therefore often correlate requests based on the security context, the client credentials, or the network address. When the Echo option increases a server's ability to correlate requests, clients MAY discard all pre-emptive Echo values.

7. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

Number	Name	Reference
TBD1	Echo	[[this document]]
TBD2	Request-Tag	[[this document]]

Figure 5: CoAP Option Numbers

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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/info/rfc7959>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP",
draft-ietf-core-oscore-groupcomm-05 (work in progress),
July 2019.
- [I-D.ietf-core-stateless]
Hartke, K., "Extended Tokens and Stateless Clients in the
Constrained Application Protocol (CoAP)", draft-ietf-core-
stateless-03 (work in progress), October 2019.

- [I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F.,
and C. Amsuess, "Controlling Actuators with CoAP", draft-
mattsson-core-coap-actuators-06 (work in progress),
September 2018.
- [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/info/rfc6347>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for
the Constrained Application Protocol (CoAP)", RFC 7390,
DOI 10.17487/RFC7390, October 2014,
<<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641,
DOI 10.17487/RFC7641, September 2015,
<<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K.,
Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained
Application Protocol) over TCP, TLS, and WebSockets",
RFC 8323, DOI 10.17487/RFC8323, February 2018,
<<https://www.rfc-editor.org/info/rfc8323>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol
Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
<<https://www.rfc-editor.org/info/rfc8446>>.
- [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/info/rfc8613>>.

Appendix A. Methods for Generating Echo Option Values

The content and structure of the Echo option value are implementation specific and determined by the server. Two simple mechanisms for time-based freshness are outlined in this section, the first is RECOMMENDED in general, and the second is RECOMMENDED in case the Echo option is encrypted between the client and the server.

Different mechanisms have different tradeoffs between the size of the Echo option value, the amount of server state, the amount of computation, and the security properties offered. A server MAY use different methods and security levels for different uses cases

(client aliveness, request freshness, state synchronization, network address reachability, etc.).

1. List of Cached Random Values and Timestamps. The Echo option value is a (pseudo-)random byte string. The server caches a list containing the random byte strings and their transmission times. Assuming 72-bit random values and 32-bit timestamps, the size of the Echo option value is 9 bytes and the amount of server state is $13n$ bytes, where n is the number of active Echo Option values. The security against an attacker guessing echo values is given by $s = \text{bit length of } r - \log_2(n)$. The length of r and the maximum allowed n should be set so that the security level is harmonized with other parts of the deployment, e.g., $s \geq 64$. If the server loses time continuity, e.g. due to reboot, the entries in the old list MUST be deleted.

Echo option value: random value r
Server State: random value r , timestamp t_0

2. Integrity Protected Timestamp. The Echo option value is an integrity protected timestamp. The timestamp can have different resolution and range. A 32-bit timestamp can e.g. give a resolution of 1 second with a range of 136 years. The (pseudo-)random secret key is generated by the server and not shared with any other party. The use of truncated HMAC-SHA-256 is RECOMMENDED. With a 32-bit timestamp and a 64-bit MAC, the size of the Echo option value is 12 bytes and the Server state is small and constant. The security against an attacker guessing echo values is given by the MAC length. If the server loses time continuity, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. Note that the privacy considerations in Section 6 may apply to the timestamp. A server MAY want to encrypt its timestamps, and, depending on the choice of encryption algorithms, this may require a nonce to be included in the Echo option value.

Echo option value: timestamp t_0 , $\text{MAC}(k, t_0)$
Server State: secret key k

Other mechanisms complying with the security and privacy considerations may be used. The use of encrypted timestamps in the Echo option increases security, but typically requires an IV to be included in the Echo option value, which adds overhead and makes the specification of such a mechanism slightly more complicated than the two mechanisms specified here.

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.5.1) incurs no overhead if no messages are lost (more precisely: in OSCORE, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when block-wise request operations happen rarely (in OSCORE, if there is always only one request block-wise operation in the replay window).

In those situations, no message has any Request-Tag option set, and that can be recycled indefinitely.

When the absence of a Request-Tag option can not be recycled any more within a security context, the messages with a present but empty Request-Tag option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) are available.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

- o In DTLS, a new session can be established.
- o In OSCORE, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Appendix C. Change Log

[The editor is asked to remove this section before publication.]

- o Changes since draft-ietf-core-echo-request-tag-07 (largely addressing Francesca's review):
 - * Request tag: Explicitly limit "MUST NOT recycle" requirement to particular applications
 - * Token reuse: upper-case RECOMMEND sequence number approach
 - * Structure: Move per-topic introductions to respective chapters (this avoids long jumps by the reader)
 - * Structure: Group Block2 / ETag section inside new fragmentation (formerly Request-Tag) section

- * More precise references into other documents
- * "concurrent operations": Emphasise that all here only matters between endpoint pairs
- * Freshness: Generalize wording away from time-based freshness
- * Echo: Emphasise that no binding between any particular pair of responses and requests is established
- * Echo: Add event-based example
- * Echo: Clarify when protection is needed
- * Request tag: Enhance wording around "not sufficient condition"
- * Request tag: Explicitly state when a tag needs to be set
- * Request tag: Clarification about permissibility of leaving the option absent
- * Security considerations: wall clock time -> system time (and remove inaccurate explanations)
- * Token reuse: describe blacklisting in a more implementation-independent way
- o Changes since draft-ietf-core-echo-request-tag-06:
 - * Removed visible comment that should not be visible in Token reuse considerations.
- o Changes since draft-ietf-core-echo-request-tag-05:
 - * Add privacy considerations on cookie-style use of Echo values
 - * Add security considerations for token reuse
 - * Add note in security considerations on use of nonvolatile memory when dealing with pseudorandom numbers
 - * Appendix on echo generation: add a few words on up- and downsides of the encrypted timestamp alternative
 - * Clarifications around Outer Echo:
 - + Could be generated by the origin server to prove network reachability (but for most applications it MUST be inner)

- + Could be generated by intermediaries
- + Is answered by the client to the endpoint from which it received it (ie. Outer if received as Outer)
- * Clarification that a server can send Echo preemptively
- * Refer to stateless to explain what "more information than just the sequence number" could be
- * Remove explanations around 0.00 empty messags
- * Rewordings:
 - + the attack: from "forging" to "guessing"
 - + "freshness tokens" to "freshness indicators" (to avoid confusion with the Token)
- * Editorial fixes:
 - + Abstract and introduction mention what is updated in RFC7252
 - + Reference updates
 - + Capitalization, spelling, terms from other documents
- o Changes since draft-ietf-core-echo-request-tag-04:
 - * Editorial fixes
 - + Moved paragraph on collision-free encoding of data in the Token to Security Considerations and rephrased it
 - + "easiest" -> "one easy"
- o Changes since draft-ietf-core-echo-request-tag-03:
 - * Mention Token processing changes in title
 - * Abstract reworded
 - * Clarify updates to Token processing
 - * Describe security levels from Echo length
 - * Allow non-monotonic clocks under certain conditions for freshness

- * Simplify freshness expressions
- * Describe when a Request-Tag can be set
- * Add note on application-level freshness mechanisms
- * Minor editorial changes
- o Changes since draft-ietf-core-echo-request-tag-02:
 - * Define "freshness"
 - * Note limitations of "aliveness"
 - * Clarify proxy and OSCORE handling in presence of "echo"
 - * Clarify when Echo values may be reused
 - * Update security considerations
 - * Various minor clarifications
 - * Minor editorial changes
- o Major changes since draft-ietf-core-echo-request-tag-01:
 - * Follow-up changes after the "relying on block-wise" change in -01:
 - + Simplify the description of Request-Tag and matchability
 - + Do not update RFC7959 any more
 - * Make Request-Tag repeatable.
 - * Add rationale on not relying purely on sequence numbers.
- o Major changes since draft-ietf-core-echo-request-tag-00:
 - * Reworded the Echo section.
 - * Added rules for Token processing.
 - * Added security considerations.
 - * Added actual IANA section.

- * Made Request-Tag optional and safe-to-forward, relying on block-wise to treat it as part of the cache-key
- * Dropped use case about OSCORE Outer-block-wise (the case went away when its Partial IV was moved into the Object-Security option)
- o Major changes since draft-amsuess-core-repeat-request-tag-00:
 - * The option used for establishing freshness was renamed from "Repeat" to "Echo" to reduce confusion about repeatable options.
 - * The response code that goes with Echo was changed from 4.03 to 4.01 because the client needs to provide better credentials.
 - * The interaction between the new option and (cross) proxies is now covered.
 - * Two messages being "Request-Tag matchable" was introduced to replace the older concept of having a request tag value with its slightly awkward equivalence definition.

Acknowledgments

The authors want to thank Carsten Bormann, Francesca Palombini, and Jim Schaad for providing valuable input to the draft.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Preuss Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

CORE
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2020

M. Boucadair
Orange
T. Reddy
McAfee
J. Shallow
October 17, 2019

Constrained Application Protocol (CoAP) Hop-Limit Option
draft-ietf-core-hop-limit-07

Abstract

The presence of Constrained Application Protocol (CoAP) proxies may lead to infinite forwarding loops, which is undesirable. To prevent and detect such loops, this document specifies the Hop-Limit CoAP option.

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 April 19, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Intended Usage	2
2. Terminology	3
3. Hop-Limit Option	3
4. Debugging & Troubleshooting	5
5. HTTP-Mapping Considerations	5
6. IANA Considerations	6
6.1. CoAP Response Code	6
6.2. CoAP Option Number	6
7. Security Considerations	7
8. Acknowledgements	7
9. References	7
9.1. Normative References	8
9.2. Informative References	8
Authors' Addresses	8

1. Introduction

More and more applications are using the Constrained Application Protocol (CoAP) [RFC7252] as a communication protocol between application agents. For example, [I-D.ietf-dots-signal-channel] specifies how CoAP is used as a signaling protocol between domains under distributed denial-of-service (DDoS) attacks and DDoS mitigation providers. In such contexts, a CoAP client can communicate directly with a server or indirectly via proxies.

When multiple proxies are involved, infinite forwarding loops may be experienced (e.g., routing misconfiguration, policy conflicts). To prevent such loops, this document defines a new CoAP option, called Hop-Limit (Section 3). Also, the document defines a new CoAP Response Code (Section 6.1) to report loops together with relevant diagnostic information to ease troubleshooting (Section 4).

1.1. Intended Usage

The Hop-Limit option was originally designed for a specific use case [I-D.ietf-dots-signal-channel]. However, its intended usage is general:

New CoAP proxies MUST implement this option and have it enabled by default.

Note that this means that a server that receives requests both via proxies and directly from clients may see otherwise identical requests with and without the Hop-Limit option included; servers with internal caching will therefore also want to implement this option, since understanding the Hop-Limit option will improve caching efficiency.

2. Terminology

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.

Readers should be familiar with the terms and concepts defined in [RFC7252].

3. Hop-Limit Option

The properties of the Hop-Limit option are shown in Table 1. The formatting of this table follows the one used in Table 4 of [RFC7252] (Section 5.10). The C, U, N, and R columns indicate the properties Critical, Unsafe, NoCacheKey, and Repeatable defined in Section 5.4 of [RFC7252]. None of these properties is marked for the Hop-Limit option.

Number	C	U	N	R	Name	Format	Length	Default
TBA2					Hop-Limit	uint	1	16

Table 1: CoAP Hop-Limit Option Properties

The Hop-Limit option (Section 6.2) is an elective option used to detect and prevent infinite loops of CoAP requests when proxies are involved. The option is not repeatable. Therefore, any request carrying multiple Hop-Limit options MUST be handled following the procedure specified in Section 5.4.5 of [RFC7252].

The value of the Hop-Limit option is encoded as an unsigned integer (see Section 3.2 of [RFC7252]). This value MUST be between 1 and 255 inclusive. CoAP requests received with a Hop-Limit option set to '0' or greater than '255' MUST be rejected by a CoAP server/proxy using 4.00 (Bad Request).

The Hop-Limit option is safe to forward. That is, a CoAP proxy that does not understand the Hop-Limit option should forward it on. The option is also part of the cache key. As such, a CoAP proxy that does not understand the Hop-Limit option must follow the recommendations in Section 5.7.1 of [RFC7252] for caching. Note that loops that involve only such proxies will not be detected. Nevertheless, the presence of such proxies will not prevent infinite loop detection if at least one CoAP proxy that supports the Hop-Limit option is involved in the loop.

A CoAP proxy that understands the Hop-Limit option SHOULD be instructed, using a configuration parameter, to insert a Hop-Limit option when relaying a request that does not include the Hop-Limit option.

The initial Hop-Limit value should be configurable. If no initial value is explicitly provided, the default initial Hop-Limit value of 16 MUST be used. This value is chosen so that in the majority of cases it is sufficiently large to guarantee that a CoAP request would not be dropped in networks when there were no loops, but not so large as to consume CoAP proxy resources when a loop does occur. The value is still configurable to accommodate unusual topologies. Lower values should be used with caution and only in networks where topologies are known by the CoAP client (or proxy) inserting the Hop-Limit option.

Because forwarding errors may occur if inadequate Hop-Limit values are used, proxies at the boundaries of an administrative domain MAY be instructed to remove or rewrite the value of Hop-Limit carried in received requests (i.e., ignore the value of Hop-Limit received in a request). This modification should be done with caution in case proxy-forwarded traffic repeatedly crosses the administrative domain boundary in a loop rendering ineffective the efficacy of loop detection through the Hop-Limit option.

Otherwise, a CoAP proxy that understands the Hop-Limit option MUST decrement the value of the option by 1 prior to forwarding it. A CoAP proxy that understands the Hop-Limit option MUST NOT use a stored TBA1 (Hop Limit Reached) error response unless the value of the Hop-Limit option in the presented request is smaller than or equal to the value of the Hop-Limit option in the request used to obtain the stored response. Otherwise, the CoAP proxy follows the behavior in Section 5.6 of [RFC7252].

Note: If a request with a given value of Hop-Limit failed to reach a server because the hop limit is exhausted, then the same failure will be observed if a smaller value of the Hop-Limit option is used instead.

CoAP requests MUST NOT be forwarded if the Hop-Limit option is set to '0' after decrement. Requests that cannot be forwarded because of exhausted Hop-Limit SHOULD be logged with a TBA1 (Hop Limit Reached) error response sent back to the CoAP peer. It is RECOMMENDED that CoAP implementations support means to alert administrators about loop errors so that appropriate actions are undertaken.

4. Debugging & Troubleshooting

To ease debugging and troubleshooting, the CoAP proxy that detects a loop includes an identifier for itself in the diagnostic payload under the conditions detailed in Section 5.5.2 of [RFC7252]. That identifier MUST NOT include any space character (ASCII value 32). The identifier inserted by a CoAP proxy can be, for example, a proxy name (e.g., p11.example.net), proxy alias (e.g., myproxyalias), or IP address (e.g., 2001:db8::1).

Each intermediate proxy involved in relaying a TBA1 (Hop Limit Reached) error message prepends its own identifier in the diagnostic payload with a space character used as separator. Only one identifier per proxy should appear in the diagnostic payload. This approach allows to limit the size of the TBA1 (Hop Limit Reached) error message, ease correlation with hops count, and detect whether a proxy was involved in the forwarding of the TBA1 (Hop Limit Reached) error message. Note that an intermediate proxy prepends its identifier only if there is enough space as determined by the Path MTU (Section 4.6 of [RFC7252]). If not, an intermediate proxy forwards the TBA1 (Hop Limit Reached) error message to the next hop without updating the diagnostic payload.

An intermediate proxy MUST NOT forward a TBA1 (Hop Limit Reached) error message if it detects that its identifier is included in the diagnostic payload. Such messages SHOULD be logged and appropriate alerts sent to the administrators.

5. HTTP-Mapping Considerations

This section focuses on the HTTP mappings specific to the CoAP extension specified in this document. As a reminder, the basic normative requirements on HTTP/CoAP mappings are defined in Section 10 of [RFC7252]. The implementation guidelines for HTTP/CoAP mappings are elaborated in [RFC8075].

By default, the HTTP-to-CoAP Proxy inserts a Hop-Limit option following the guidelines in Section 3. The HTTP-to-CoAP Proxy may be instructed by policy to insert a Hop-Limit option only if a Via (Section 5.7.1 of [RFC7230]) or CDN-Loop header field [RFC8586] is present in the HTTP request.

The HTTP-to-CoAP Proxy uses 508 (Loop Detected) as the HTTP response status code to map TBA1 (Hop Limit Reached). Furthermore, it maps the diagnostic payload of TBA1 (Hop Limit Reached) as per Section 6.6 of [RFC8075].

By default, the CoAP-to-HTTP Proxy inserts a Via header field in the HTTP request if the CoAP request includes a Hop-Limit option. The CoAP-to-HTTP Proxy may be instructed by policy to insert a CDN-Loop header field instead of the Via header field.

The CoAP-to-HTTP Proxy maps the 508 (Loop Detected) HTTP response status code to TBA1 (Hop Limit Reached). Moreover, the CoAP-to-HTTP Proxy inserts its information following the guidelines in Section 4.

When both HTTP-to-CoAP and CoAP-to-HTTP proxies are involved, the loop detection may get broken if the proxy-forwarded traffic repeatedly crosses the HTTP-to-CoAP and CoAP-to-HTTP proxies. Nevertheless, if the loop is within the CoAP or HTTP legs, the loop detection is still functional.

6. IANA Considerations

Editorial Note: Please update TBA1/TBA2 statements within the document with the assigned codes.

6.1. CoAP Response Code

IANA is requested to add the following entry to the "CoAP Response Codes" sub-registry available at <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#response-codes>:

Code	Description	Reference
TBA1	Hop Limit Reached	[RFCXXXX]

Table 2: CoAP Response Codes

This document suggests 5.08 as a code to be assigned for the new response code.

6.2. CoAP Option Number

IANA is requested to add the following entry to the "CoAP Option Numbers" sub-registry available at <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#option-numbers>:

Number	Name	Reference
TBA2	Hop-Limit	[RFCXXXX]

Table 3: CoAP Option Number

This document suggests 16 as a value to be assigned for the new option number.

7. Security Considerations

Security considerations related to CoAP proxying are discussed in Section 11.2 of [RFC7252].

A CoAP endpoint can probe the topology of a network into which it is making requests by tweaking the value of the Hop-Limit option. Such probing is likely to fail if proxies at the boundaries of that network rewrite the value of Hop-Limit carried in received requests (see Section 3).

The diagnostic payload of a TBA1 (Hop Limit Reached) error message may leak sensitive information revealing the topology of an administrative domain. To prevent that, a CoAP proxy that is located at the boundary of an administrative domain MAY be instructed to strip the diagnostic payload or part of it before forwarding on the TBA1 (Hop Limit Reached) response.

8. Acknowledgements

This specification was part of [I-D.ietf-dots-signal-channel]. Many thanks to those who reviewed DOTS specifications.

Thanks to Klaus Hartke, Carsten Bormann, Peter van der Stok, Jim Schaad, Jaime Jimenez, Roni Even, Scott Bradner, Thomas Fossati, Radia Perlman, Eric Vyncke, Suresh Krishnan, Roman Danyliw, Barry Leiba, Christer Holmberg, Benjamin Kaduk, and Adam Roach for their review and comments.

Carsten Bormann provided the "Intended Usage" text.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [I-D.ietf-dots-signal-channel] K, R., Boucadair, M., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", draft-ietf-dots-signal-channel-37 (work in progress), July 2019.
- [RFC8586] Ludin, S., Nottingham, M., and N. Sullivan, "Loop Detection in Content Delivery Networks (CDNs)", RFC 8586, DOI 10.17487/RFC8586, April 2019, <<https://www.rfc-editor.org/info/rfc8586>>.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Tirumaleswar Reddy
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore, Karnataka 560071
India

Email: kondtir@gmail.com

Jon Shallow
United Kingdom

Email: supjps-ietf@jpshallow.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

K. Hartke
Ericsson
November 4, 2019

Constrained Resource Identifiers
draft-ietf-core-href-01

Abstract

Constrained Resource Identifiers (CoRIs) are an alternate serialization of Uniform Resource Identifiers (URIs) that encodes the URI components in Concise Binary Object Representation (CBOR) instead of a string of characters. This simplifies parsing, reference resolution, and comparison of URIs in environments with severe limitations on processing power, code size, and memory size.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. Data Model	3
2.1. Options	3
2.2. Option Sequences	4
3. CBOR	7
4. Python	7
4.1. Reference Resolution	9
4.2. URI Recomposition	10
4.3. CoAP Encoding	12
5. Security Considerations	14
6. IANA Considerations	14
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Acknowledgements	15
Author's Address	15

1. Introduction

Uniform Resource Identifier (URI) references [RFC3986] are the standard way to link to resources in hypertext formats such as HTML [W3C.REC-html52-20171214] or the HTTP "Link" header field [RFC8288]. A URI reference is either a URI or a relative reference that must be resolved against a base URI.

URI references are strings of characters chosen from the repertoire of US-ASCII characters. The individual components of a URI reference are delimited by a number of reserved characters, which necessitates the use of percent-encoding when these reserved characters are used in a non-delimiting function. One component can also contain special dot-segments that affect how the component is to be interpreted. The resolution of URI references involves parsing the character string into its components, combining those components with the components of a base URI, merging path components, removing dot-segments, and recomposing the result back into a character string.

Overall, the proper processing of URIs is quite complicated. This can be a problem in particular in constrained environments [RFC7228], where devices often have severe code size limitations. As a result, many implementations in these environments choose to support only an ad-hoc, informally-specified, bug-ridden, non-interoperable subset of half of the URI standard.

This document introduces Constrained Resource Identifier (CoRI) references, an alternate serialization of URI references that encodes the URI components in Concise Binary Object Representation (CBOR) [RFC7049] instead of a string of characters. Assuming an implementation of CBOR is already present on a device, typical operations on URI references such as parsing, reference resolution, and comparison can be implemented more easily than for character strings. A full implementation that covers all corner cases is intended to be implementable in a relatively small amount of code.

As a result of the simplification, CoRI references are not capable of expressing all URI references permitted by the syntax of RFC 3986. (Hence the "constrained" in "Constrained Resource Identifiers".) The supported subset includes all Constrained Application Protocol (CoAP) URIs [RFC7252], most Hypertext Transfer Protocol (HTTP) URIs [RFC7230], and many other URIs that function as resource locators.

1.1. Notational Conventions

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.

Terms defined in this document appear in *_cursive_* where they are introduced.

2. Data Model

The data model for CoRI references is very similar to the serialization of the request URI in CoAP messages [RFC7252]: The components of a URI reference are encoded as a sequence of *_options_*, where each path segment and query parameter becomes its own option. Every option consists of an *_option number_* identifying the type of option (scheme, host name, path segment, etc.) and an *_option value_*.

2.1. Options

The following types of options are defined:

scheme

Specifies the URI scheme. The option value can be any Unicode string matching the "scheme" rule described in Section 3.1 of RFC 3986 [RFC3986].

host.name

Specifies the host of the URI authority as a registered name. The option value can be any Unicode string matching the specifications of the URI scheme.

host.ip

Specifies the host of the URI authority as an IPv4 address or an IPv6 address. The option value is a byte string with a length of either 4 or 16 bytes, respectively.

port

Specifies the port number of the URI authority. The option value is an integer in the range from 0 to 65535.

path.type

Specifies the type of the URI path for reference resolution. The option value is an integer in the range from 0 to 127, named as follows:

- 0 - absolute-path
- 1 - append-relation
- 2 - append-path
- 3 - relative-path
- 4 - relative-path-1up
- 5 - relative-path-2up
- 6 - relative-path-3up
- 7 - relative-path-4up
- ...

path

Specifies one segment of the URI path. The option value can be any Unicode string with the exception of "." and "..". This option can occur more than once.

query

Specifies one argument of the URI query. The option value can be any Unicode string. This option can occur more than once.

fragment

Specifies the fragment identifier. The option value can be any Unicode string.

No percent-encoding is performed in option values.

2.2. Option Sequences

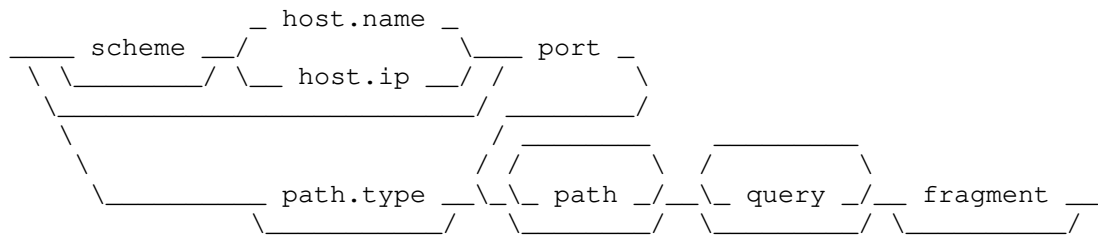


Figure 1: Structure of a Well-Formed Sequence of Options

A sequence of options is considered well-formed if:

- o the sequence of options is empty or starts with a "scheme", "host.name", "host.ip", "port", "path.type", "path", "query", or "fragment" option;
- o any "scheme" option is followed by either a "host.name" or a "host.ip" option;
- o any "host.name" option is followed by a "port" option;
- o any "host.ip" option is followed by a "port" option;
- o any "port" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;
- o any "path.type" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;
- o any "path" option is followed by a "path", "query", or "fragment" option or is at the end of the sequence;
- o any "query" option is followed by a "query" or "fragment" option or is at the end of the sequence; and
- o any "fragment" option is at the end of the sequence.

A well-formed sequence of options is considered absolute if the sequence of options starts with a "scheme" option.

A well-formed sequence of options is considered relative if the sequence of options is empty or starts with an option other than a "scheme" option.

An absolute sequence of options is considered normalized if the result of resolving the sequence of options against any base is equal

to the input. (It doesn't matter what base it is resolved against, since it is already absolute.)

The following operations can be performed on a sequence of options:

`resolve(href, base)`

Resolves a well-formed sequence of options `'href'` against an absolute sequence of options `'base'`. This operation **MUST** be performed by applying any algorithm that is functionally equivalent to the reference implementation in Section 4.1 of this document.

`relative(href, base)`

Makes an absolute sequence of options `'href'` relative to an absolute sequence of options `'base'`. This operation **MUST** be performed by applying any algorithm that returns a sequence of options such that `'resolve(relative(h, b), b)'` is equal to `'h'` given the same `'b'`.

`recompose(href)`

Recomposes a URI from an absolute sequence of options `'href'`. This operation **MUST** be performed by applying any algorithm that is functionally equivalent to the reference implementation in Section 4.2 of this document.

To reduce variability, it is **RECOMMENDED** to uppercase the letters in the hexadecimal notation when percent-encoding octets [RFC3986] and to follow the recommendations of Section 4 of RFC 5952 for the text representation of IPv6 addresses [RFC5952].

`decompose(str)`

Decomposes a URI `'str'` into a sequence of options. This operation **MUST** be performed by applying any algorithm that returns a sequence of options such that `'recompose(decompose(x))'` is equivalent to `'x'`.

`coap(href)`

Constructs CoAP options from an absolute, normalized sequence of options. This operation **MUST** be performed by recomposing the sequence of options to a URI (as described above) and decomposing the URI into CoAP options (as specified in Section 6.4 of RFC 7252). A concise implementation of this algorithm is illustrated in Section 4.3 of this document.

3. CBOR

In Concise Binary Object Representation (CBOR) [RFC7049], a sequence of options is encoded as an array that contains the option numbers and option values in alternating order.

The structure can be described in the Concise Data Definition Language (CDDL) [RFC8610] as follows:

```
CoRI = [?(scheme: 1, text .regexp "[A-Za-z][A-Za-z0-9+.-]*"),
        ?(host.name: 2, text //
          host.ip: 3, bytes .size 4 / bytes .size 16),
        ?(port: 4, 0..65535),
        ?(path.type: 5, 0..127),
        *(path: 6, text),
        *(query: 7, text),
        ?(fragment: 8, text)]
```

Example:

```
[1, "coap", 3, h'20010DB8000000000000000000000001', 4, 5683, 6,
 ".well-known", 6, "core"]

[5, 0, 6, ".well-known", 6, "core", 7, "rt=temperature-c"]
```

4. Python

In Python, a sequence of options is encoded as a list of tuples, where each tuple contains one option number and one option value.

The following Python 3.6 code illustrates how to check a sequence of options for being well-formed, absolute, and relative.

<CODE BEGINS>

```
import enum

class Option(enum.IntEnum):
    _BEGIN = 0
    SCHEME = 1
    HOST_NAME = 2
    HOST_IP = 3
    PORT = 4
    PATH_TYPE = 5
    PATH = 6
    QUERY = 7
    FRAGMENT = 8
    _END = 9
```

```

class PathType(enum.IntEnum):
    ABSOLUTE_PATH = 0
    APPEND_RELATION = 1
    APPEND_PATH = 2
    RELATIVE_PATH = 3
    RELATIVE_PATH_1UP = 4
    RELATIVE_PATH_2UP = 5
    RELATIVE_PATH_3UP = 6
    RELATIVE_PATH_4UP = 7

_TRANSITIONS = ([Option.SCHEME, Option.HOST_NAME, Option.HOST_IP,
    Option.PORT, Option.PATH_TYPE, Option.PATH, Option.QUERY,
    Option.FRAGMENT, Option._END],
    [Option.HOST_NAME, Option.HOST_IP],
    [Option.PORT],
    [Option.PORT],
    [Option.PATH, Option.QUERY, Option.FRAGMENT, Option._END],
    [Option.PATH, Option.QUERY, Option.FRAGMENT, Option._END],
    [Option.PATH, Option.QUERY, Option.FRAGMENT, Option._END],
    [Option.QUERY, Option.FRAGMENT, Option._END],
    [Option._END])

def is_well_formed(href):
    previous = Option._BEGIN
    for option, _ in href:
        if option not in _TRANSITIONS[previous]:
            return False
        previous = option
    if Option._END not in _TRANSITIONS[previous]:
        return False
    return True

def is_absolute(href):
    return is_well_formed(href) and \
        (len(href) != 0 and href[0][0] == Option.SCHEME)

def is_relative(href):
    return is_well_formed(href) and \
        (len(href) == 0 or href[0][0] != Option.SCHEME)

<CODE ENDS>

```

Examples:

```

[(Option.SCHEME, "coap"), (Option.HOST_IP, b"\x20\x01\x0D\xB8\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01"), (Option.PORT, 5683), (Option.PATH, ".well-known"), (Option.PATH, "core")]

```

```
[(Option.PATH_TYPE, PathType.ABSOLUTE_PATH), (Option.PATH, ".well-known"), (Option.PATH, "core"), (Option.QUERY, "rt=temperature-c")]
```

4.1. Reference Resolution

The following Python 3.6 code defines how to resolve a sequence of options that might be relative to a given base.

<CODE BEGINS>

```
def resolve(base, href, relation=0):
    if not is_absolute(base) or not is_well_formed(href):
        return None
    result = []
    option = Option.FRAGMENT
    if len(href) != 0:
        option = href[0][0]
    if option == Option.HOST_IP:
        option = Option.HOST_NAME
    elif option == Option.PATH_TYPE:
        type = href[0][1]
        href = href[1:]
    elif option == Option.PATH:
        type = PathType.RELATIVE_PATH
        option = Option.PATH_TYPE
    if option != Option.PATH_TYPE or type == PathType.ABSOLUTE_PATH:
        _copy_until(base, result, option)
    else:
        _copy_until(base, result, Option.QUERY)
        if type == PathType.APPEND_RELATION:
            _append_and_normalize(result, Option.PATH, str(relation))
        while type > PathType.APPEND_PATH:
            if len(result) == 0 or result[-1][0] != Option.PATH:
                break
            del result[-1]
            type -= 1
        _copy_until(href, result, Option._END)
        _append_and_normalize(result, Option._END, None)
    return result

def _copy_until(input, output, end):
    for option, value in input:
        if option >= end:
            break
        _append_and_normalize(output, option, value)

def _append_and_normalize(output, option, value):
```



```
if option > Option.PATH:
    if len(output) >= 2 and \
        output[-1] == (Option.PATH, '') and (
            output[-2][0] < Option.PATH_TYPE or (
                output[-2][0] == Option.PATH_TYPE and
                output[-2][1] == PathType.ABSOLUTE_PATH)):
        del output[-1]
    if option > Option.FRAGMENT:
        return
    output.append((option, value))
```

<CODE ENDS>

4.2. URI Recomposition

The following Python 3.6 code defines how to recompose a URI from an absolute sequence of options.

<CODE BEGINS>

```
def recompose(href):
    if not is_absolute(href):
        return None
    result = ''
    no_path = True
    first_query = True
    for option, value in href:
        if option == Option.SCHEME:
            result += value + ':'
        elif option == Option.HOST_NAME:
            result += '//' + _encode_reg_name(value)
        elif option == Option.HOST_IP:
            result += '//' + _encode_ip_address(value)
        elif option == Option.PORT:
            result += ':' + _encode_port(value)
        elif option == Option.PATH:
            result += '/' + _encode_path_segment(value)
            no_path = False
        elif option == Option.QUERY:
            if no_path:
                result += '/'
            no_path = False
            result += '?' if first_query else '&'
            result += _encode_query_argument(value)
            first_query = False
        elif option == Option.FRAGMENT:
            if no_path:
                result += '/'
```

```
        no_path = False
        result += '#' + _encode_fragment(value)
    if no_path:
        result += '/'
        no_path = False
    return result

def _encode_reg_name(s):
    return ''.join(c if _is_reg_name_char(c)
                   else _encode_pct(c) for c in s)

def _encode_ip_address(b):
    if len(b) == 4:
        return '.'.join(str(c) for c in b)
    elif len(b) == 16:
        return '[' + ... + ']' # see RFC 5952

def _encode_port(p):
    return str(p)

def _encode_path_segment(s):
    return ''.join(c if _is_segment_char(c)
                   else _encode_pct(c) for c in s)

def _encode_query_argument(s):
    return ''.join(c if _is_query_char(c) and c not in '&'
                   else _encode_pct(c) for c in s)

def _encode_fragment(s):
    return ''.join(c if _is_fragment_char(c)
                   else _encode_pct(c) for c in s)

def _encode_pct(s):
    return ''.join('%{0:0>2X}'.format(c) for c in s.encode('utf-8'))

def _is_reg_name_char(c):
    return _is_unreserved(c) or _is_sub_delim(c)

def _is_segment_char(c):
    return _is_pchar(c)

def _is_query_char(c):
    return _is_pchar(c) or c in '/?'

def _is_fragment_char(c):
    return _is_pchar(c) or c in '/?'

def _is_pchar(c):
```

```
    return _is_unreserved(c) or _is_sub_delim(c) or c in ':@'

def _is_unreserved(c):
    return _is_alpha(c) or _is_digit(c) or c in '-._~'

def _is_alpha(c):
    return c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' + \
        'abcdefghijklmnopqrstuvwxyz'

def _is_digit(c):
    return c in '0123456789'

def _is_sub_delim(c):
    return c in '!$%&\'()*+,-;='
```

<CODE ENDS>

4.3. CoAP Encoding

The following Python 3.6 code illustrates how to construct CoAP options from an absolute sequence of options. For simplicity, the code does not omit CoAP options with their default value.

<CODE BEGINS>

```
def coap(href, to_proxy=False):
    if not is_absolute(href):
        return None
    result = b''
    previous = 0
    for option, value in href:
        if option == Option.SCHEME:
            pass
        elif option == Option.HOST_NAME:
            opt = 3 # Uri-Host
            val = value.encode('utf-8')
            result += _encode_coap_option(opt - previous, val)
            previous = opt
        elif option == Option.HOST_IP:
            opt = 3 # Uri-Host
            if len(value) == 4:
                val = '.'.join(str(c) for c in value).encode('utf-8')
            elif len(value) == 16:
                val = b'[' + ... + b']' # see RFC 5952
            result += _encode_coap_option(opt - previous, val)
            previous = opt
        elif option == Option.PORT:
            opt = 7 # Uri-Port
```

```
        val = value.to_bytes((value.bit_length() + 7) // 8, 'big')
        result += _encode_coap_option(opt - previous, val)
        previous = opt
    elif option == Option.PATH:
        opt = 11 # Uri-Path
        val = value.encode('utf-8')
        result += _encode_coap_option(opt - previous, val)
        previous = opt
    elif option == Option.QUERY:
        opt = 15 # Uri-Query
        val = value.encode('utf-8')
        result += _encode_coap_option(opt - previous, val)
        previous = opt
    elif option == Option.FRAGMENT:
        pass
    if to_proxy:
        (option, value) = href[0]
        opt = 39 # Proxy-Scheme
        val = value.encode('utf-8')
        result += _encode_coap_option(opt - previous, val)
        previous = opt
    return result

def _encode_coap_option(delta, value):
    length = len(value)
    delta_nibble = _encode_coap_option_nibble(delta)
    length_nibble = _encode_coap_option_nibble(length)
    result = bytes([delta_nibble << 4 | length_nibble])
    if delta_nibble == 13:
        delta -= 13
        result += bytes([delta])
    elif delta_nibble == 14:
        delta -= 256 + 13
        result += bytes([delta >> 8, delta & 255])
    if length_nibble == 13:
        length -= 13
        result += bytes([length])
    elif length_nibble == 14:
        length -= 256 + 13
        result += bytes([length >> 8, length & 255])
    result += value
    return result

def _encode_coap_option_nibble(n):
    if n < 13:
        return n
    elif n < 256 + 13:
        return 13
```

```
elif n < 65536 + 256 + 13:  
    return 14
```

<CODE ENDS>

5. Security Considerations

Parsers must operate on input that is assumed to be untrusted. This means that parsers **MUST** fail gracefully in the face of malicious inputs. Additionally, parsers **MUST** be prepared to deal with resource exhaustion (e.g., resulting from the allocation of big data items) or exhaustion of the call stack (stack overflow). See Section 8 of RFC 7049 [RFC7049] for security considerations relating to CBOR.

The security considerations discussed in Section 7 of RFC 3986 [RFC3986] also apply to Constrained Resource Identifiers.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

7.2. Informative References

- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <<https://www.w3.org/TR/2017/REC-html52-20171214>>.

Acknowledgements

Thanks to Christian Amsuess, Ari Keranen, and Dave Thaler for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: February 22, 2020

T. Fossati
ARM
K. Hartke
Ericsson
C. Bormann
Universitaet Bremen TZI
August 21, 2019

Multipart Content-Format for CoAP
draft-ietf-core-multipart-ct-04

Abstract

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of zero or more different media types into a single message, such as a CoAP request or response body, with minimal framing overhead, each along with a CoAP Content-Format identifier.

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 February 22, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Multipart Content-Format Encoding	4
3. Usage Example: Observing Resources	4
4. Implementation Hints	5
5. IANA Considerations	6
5.1. Registration of media type application/multipart-core . .	6
5.2. Registration of a Content-Format identifier for application/multipart-core	7
6. Security Considerations	8
7. References	8
7.1. Normative References	8
7.2. Informative References	8
Acknowledgements	9
Authors' Addresses	9

1. Introduction

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of zero or more different media types, each along with a CoAP Content-Format identifier, into a single representation, with minimal framing overhead. This combined representation may then be carried in a single message, such as a CoAP [RFC7252] request or response body.

This simple and efficient binary framing mechanism can be employed to create application specific request and response bodies which build on multiple already existing media types.

As the name of the media-type suggests, it is inspired by the multipart media types that started to be defined with the original set of MIME specifications [RFC2046]. However, while those needed to focus on the syntactic aspects of integrating multiple representations into one e-mail, transfer protocols providing full data transparency such as CoAP as well as readily available encoding formats such as the Concise Binary Object Representation (CBOR) [RFC7049] shift the focus towards the intended use of the combined representations. In this respect, the basic intent of the application/multipart-core media type is like that of multipart/mixed (Section 5.1.3 of [RFC2046]). The detailed semantics of the representations are refined by the context established by the application in the accompanying request parameters, e.g., the

resource URI and any further options (header fields), but three usage scenarios are envisioned:

The individual representations in an application/multipart-core body occur in a sequence, which may be employed by an application where such a sequence is natural, e.g. for a number of audio snippets in various formats to be played out in that sequence, or search results returned in order of relevance.

In other cases, an application may be more interested in a bag of representations, which are distinguished by their Content-Format identifier, such as an audio snippet and a text representation accompanying it. In such a case, the sequence in which these occur may not be relevant to the application. This specification adds the option of substituting a null value for the representation of an optional part, which indicates that the part is not present.

A third situation that is common only ever has a single representation in the sequence, where the sender already selects just one of a set of formats possible for this situation. This kind of union "type" of formats may also make the presence of the actual representation optional, the omission of which leads to a zero-length array.

Where these rules are not sufficient for an application, it might still use the general format defined here, but register a new media type and an associated Content-Format identifier to associate the representation with these more specific semantics instead of using the application/multipart-core media type.

Also, future specifications might want to define rough equivalents for other multipart media types with specific semantics not covered by the present specification, such as multipart/alternative (Section 5.1.4 of [RFC2046]), where several alternative representations are provided in the message, but only one of those is to be selected by the recipient for its use (this is less likely to be useful in a constrained environment that has facilities for pre-flight discovery).

1.1. Requirements Language

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. Multipart Content-Format Encoding

A representation of media-type application/multipart-core contains a collection of zero or more representations, each along with their respective content format.

The collection is encoded as a CBOR [RFC7049] array with an even number of elements. Counting from zero, the odd-numbered elements are a byte string containing a representation, or the value "null" if an optional part is indicated as not given. The (even-numbered) element preceding each of these is an unsigned integer specifying the content format ID of the representation following it.

For example, a collection containing two representations, one with content format ID 42 and one with content format ID 0, looks like this in CBOR diagnostic notation:

```
[42, h'0123456789abcdef', 0, h'3031323334']
```

For illustration, the structure of an application/multipart-core representation can be described by the CDDL [RFC8610] specification in Figure 1:

```
multipart-core = [* multipart-part]
multipart-part = (type: uint .size 2, part: bytes / null)
```

Figure 1: CDDL for application/multipart-core

This format is intended as a strict specification: An implementation MUST stop processing the representation if there is a CBOR well-formedness error, a deviation from the structure defined above, or any residual data left after processing the CBOR data item. (This generally means the representation is not processed at all except if some streaming processing has already happened.)

3. Usage Example: Observing Resources

This section illustrates one less obvious example for using application/multipart-core: combining it with observing a resource [RFC7641] to handle pending results.

When a client registers to observe a resource for which no representation is available yet, the server may send one or more 2.05 (Content) notifications before sending the first actual 2.05 (Content) or 2.03 (Valid) notification. A diagram depicting possible resulting sequences of notifications, identified by their respective response code, is shown in Figure 2.

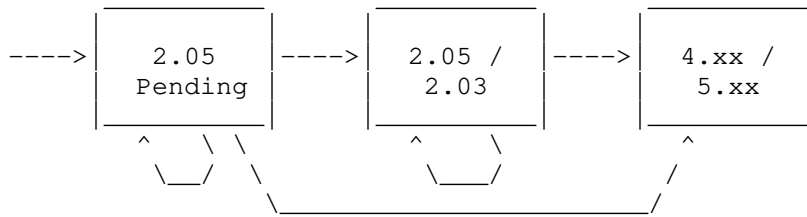


Figure 2: Sequence of Notifications

The specification of the Observe option requires that all notifications carry the same Content-Format. The application/multipart-core media type can be used to provide that Content-Format: e.g., carrying an empty list of representations in the case marked as "Pending" in Figure 2, and carrying a single representation specified as the target content-format in the case in the middle of the figure.

4. Implementation Hints

This section describes the serialization for readers that may be new to CBOR. It does not contain any new information.

An application/multipart-core representation carrying no representations is represented by an empty CBOR array, which is serialized as a single byte with the value 0x80.

An application/multipart-core representation carrying a single representation is represented by a two-element CBOR array, which is serialized as 0x82 followed by the two elements. The first element is an unsigned integer for the Content-Format value, which is represented as described in Table 1. The second element is the object as a byte string, which is represented as a length as described in Table 2 followed by the bytes of the object.

Serialization	Value
0x00..0x17	0..23
0x18 0xnn	24..255
0x19 0xnn 0xnn	256..65535

Table 1: Serialization of content-format

Serialization	Length
0x40..0x57	0..23
0x58 0xnn	24..255
0x59 0xnn 0xnn	256..65535
0x5a 0xnn 0xnn 0xnn 0xnn	65536..4294967295
0x5b 0xnn .. 0xnn (8 bytes)	4294967296..

Table 2: Serialization of object length

For example, a single text/plain object (content-format 0) of value "Hello World" (11 characters) would be serialized as

```
0x82 0x00 0x4b H e l l o 0x20 W o r l d
```

In effect, the serialization for a single object is done by prefixing the object with information that there is one object (here: 0x82), about its content-format (here: 0x00) and its length (here: 0x4b).

For more than one representation included in an application/multipart-core representation, the head of the CBOR array is adjusted (0x84 for two representations, 0x86 for three, ...) and the sequences of content-format and embedded representations follow.

For instance, the example from Section 2 would be serialized as:

```
0x84 (*) 0x182A 0x48 0x0123456789ABCDEF (+) 0x00 0x45 0x3031323334
```

where (*) marks the start of the information about the first representation (content-format 42, byte string length 8) and, (+), of the second representation (content-format 0, byte string length 5).

5. IANA Considerations

5.1. Registration of media type application/multipart-core

IANA is requested to register the following media type [RFC6838]:

Type name: application

Subtype name: multipart-core

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations Section of RFCthis

Interoperability considerations: N/A

Published specification: RFCthis

Applications that use this media type: Applications that need to combine representations of zero or more different media types into one, e.g., EST-CoAP [I-D.ietf-ace-coap-est]

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

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: CoRE WG

Change controller: IESG

Provisional registration? (standards tree only): no

5.2. Registration of a Content-Format identifier for application/multipart-core

IANA is requested to register the following Content-Format to the "CoAP Content-Formats" subregistry, within the "Constrained RESTful

Environments (CoRE) Parameters" registry, from the Expert Review space (0..255):

Media Type	Encoding	ID	Reference
application/multipart-core	--	TBD1	RFCthis

6. Security Considerations

The security considerations of [RFC7049] apply. In particular, resource exhaustion attacks may employ large values for the byte string size fields, or deeply nested structures of recursively embedded application/multipart-core representations.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

7.2. Informative References

- [I-D.ietf-ace-coap-est] Stok, P., Kampanakis, P., Richardson, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-12 (work in progress), June 2019.

- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <<https://www.rfc-editor.org/info/rfc2046>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

Acknowledgements

Most of the text in this draft is from earlier contributions by two of the authors, Thomas Fossati and Klaus Hartke. The re-mix in this document is based on the requirements in [I-D.ietf-ace-coap-est], based on discussions with Michael Richardson, Panos Kampanis and Peter van der Stok.

Authors' Addresses

Thomas Fossati
ARM

Email: thomas.fossati@arm.com

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

M. Tiloca
RISE AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
November 04, 2019

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-06

Abstract

This document describes a mode for protecting group communication over the Constrained Application Protocol (CoAP). The proposed mode relies on Object Security for Constrained RESTful Environments (OSCORE) and the CBOR Object Signing and Encryption (COSE) format. In particular, it defines how OSCORE is used in a group communication setting, while fulfilling the same security requirements for group requests and responses. Source authentication of all messages exchanged within the group is provided by means of digital signatures produced by the sender and embedded in the protected CoAP messages.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. OSCORE Security Context	5
2.1. Management of Group Keying Material	9
2.2. Wrap-Around of Partial IVs	9
3. The COSE Object	10
3.1. Updated external_aad	10
3.1.1. Updated external_aad for Encryption	10
3.1.2. Updated external_aad for Signing	11
3.2. Use of the 'kid' Parameter	12
3.3. Updated 'unprotected' Field	12
4. OSCORE Header Compression	12
4.1. Encoding of the OSCORE Option Value	12
4.2. Encoding of the OSCORE Payload	13
4.3. Examples of Compressed COSE Objects	14
5. Message Binding, Sequence Numbers, Freshness and Replay Protection	15
5.1. Synchronization of Sender Sequence Numbers	15
6. Message Processing	15
6.1. Protecting the Request	16
6.2. Verifying the Request	16
6.3. Protecting the Response	17
6.4. Verifying the Response	17
7. Responsibilities of the Group Manager	18
8. Security Considerations	19
8.1. Group-level Security	20
8.2. Uniqueness of (key, nonce)	20
8.3. Management of Group Keying Material	21
8.4. Update of Security Context and Key Rotation	21
8.5. Collision of Group Identifiers	22
8.6. Cross-group Message Injection	22
8.7. End-to-end Protection	24
8.8. Security Context Establishment	24
8.9. Master Secret	24
8.10. Replay Protection	25
8.11. Client Aliveness	25

8.12. Cryptographic Considerations	25
8.13. Message Segmentation	26
8.14. Privacy Considerations	26
9. IANA Considerations	27
9.1. Counter Signature Parameters Registry	27
9.2. Counter Signature Key Parameters Registry	29
9.3. Expert Review Instructions	31
10. References	32
10.1. Normative References	32
10.2. Informative References	33
Appendix A. Assumptions and Security Objectives	34
A.1. Assumptions	34
A.2. Security Objectives	36
Appendix B. List of Use Cases	36
Appendix C. Example of Group Identifier Format	39
Appendix D. Set-up of New Endpoints	40
Appendix E. Examples of Synchronization Approaches	40
E.1. Best-Effort Synchronization	40
E.2. Baseline Synchronization	41
E.3. Challenge-Response Synchronization	41
Appendix F. No Verification of Signatures	43
Appendix G. Document Updates	43
G.1. Version -05 to -06	43
G.2. Version -04 to -05	44
G.3. Version -03 to -04	44
G.4. Version -02 to -03	45
G.5. Version -01 to -02	46
G.6. Version -00 to -01	47
Acknowledgments	47
Authors' Addresses	48

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks [RFC7228].

Group communication for CoAP [RFC7390][I-D.dijk-core-groupcomm-bis] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies, improve performance and reduce bandwidth utilisation. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast (see Appendix B). Furthermore, [RFC7390] recognizes the importance to introduce a secure mode for CoAP group communication. This specification defines such a mode.

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC8152] and provides end-to-end encryption, integrity, replay protection and binding of response to request between a sender and a recipient, also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, providing end-to-end security of CoAP messages exchanged between members of a group, and preserving independence of transport layer. In particular, the described approach defines how OSCORE should be used in a group communication setting, so that end-to-end security is assured in the same way as OSCORE for unicast communication. That is, end-to-end security is provided for CoAP multicast requests sent by a client to the group, and for related CoAP responses sent by multiple servers. Group OSCORE provides source authentication of all CoAP messages exchanged within the group, by means of digital signatures produced through private keys of sender devices and embedded in the protected CoAP messages.

As defined in the latest [I-D.dijk-core-groupcomm-bis], Group OSCORE is the security protocol to use for applications that rely on CoAP group communication. As in OSCORE, it is still possible to simultaneously rely on DTLS [RFC6347] to protect hop-by-hop communication between a sender and a proxy (and vice versa), and between a proxy and a recipient (and vice versa). Note that DTLS cannot be used to secure messages sent over multicast.

1.1. Terminology

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.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] including "endpoint", "client", "server", "sender" and "recipient"; group communication for CoAP [RFC7390] [I-D.dijk-core-groupcomm-bis]; COSE and counter signatures [RFC8152].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such as "Security Context" and "Master Secret", defined in [RFC8613].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

This document refers also to the following terminology.

- o Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].
- o Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). The term group used in this specification refers thus to a "security group", not to be confused with network/multicast group or application group.
- o Group Manager: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager, which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 7.
- o Silent server: member of a group that never responds to requests. Note that a silent server can act as a client, the two roles are independent.
- o Group Identifier (Gid): identifier assigned to the group. Group Identifiers must be unique within the set of groups of a given Group Manager.
- o Group request: CoAP request message sent by a client in the group to all servers in that group.
- o Source authentication: evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint which is not a group member.

2. OSCORE Security Context

To support group communication secured with OSCORE, each endpoint registered as member of a group maintains a Security Context as defined in Section 3 of [RFC8613], extended as defined below. Each endpoint in a group makes use of:

1. one Common Context, shared by all the endpoints in a given group. In particular:

- * The ID Context parameter contains the Gid of the group, which is used to retrieve the Security Context for processing messages intended to the endpoints of the group (see Section 6). The choice of the Gid is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle possible collisions between Gids, see Section 8.5.
- * A new parameter Counter Signature Algorithm is included. Its value identifies the digital signature algorithm used to compute a counter signature on the COSE object (see Section 4.5 of [RFC8152]) which provides source authentication within the group. Its value is immutable once the Common Context is established. The used Counter Signature Algorithm MUST be selected among the signing ones defined in the COSE Algorithms Registry (see section 16.4 of [RFC8152]). The EdDSA signature algorithm ed25519 [RFC8032] is mandatory to implement. If Elliptic Curve Digital Signature Algorithm (ECDSA) is used, it is RECOMMENDED that implementations implement "deterministic ECDSA" as specified in [RFC6979].
- * A new parameter Counter Signature Parameters is included. This parameter identifies the parameters associated to the digital signature algorithm specified in the Counter Signature Algorithm. This parameter MAY be empty and is immutable once the Common Context is established. The exact structure of this parameter depends on the value of Counter Signature Algorithm, and is defined in the Counter Signature Parameters Registry (see Section 9.1), where each entry indicates a specified structure of the Counter Signature Parameters.
- * A new parameter Counter Signature Key Parameters is included. This parameter identifies the parameters associated to the keys used with the digital signature algorithm specified in the Counter Signature Algorithm. This parameter MAY be empty and is immutable once the Common Context is established. The exact structure of this parameter depends on the value of Counter Signature Algorithm, and is defined in the Counter Signature Key Parameters Registry (see Section 9.2), where each entry indicates a specified structure of the Counter Signature Key Parameters.

2. one Sender Context, unless the endpoint is configured exclusively as silent server. The Sender Context is used to secure outgoing messages and is initialized according to Section 3 of [RFC8613],

once the endpoint has joined the group. The Sender Context of a given endpoint matches the corresponding Recipient Context in all the endpoints receiving a protected message from that endpoint. Besides, in addition to what is defined in [RFC8613], the Sender Context stores also the endpoint's private key.

3. one Recipient Context for each distinct endpoint from which messages are received, used to process incoming messages. The recipient may generate a Recipient Context whenever in possession of all the required pieces of information on the corresponding endpoint, e.g. they may be provided to the recipient upon joining the group. Alternatively, the recipient may generate a Recipient Context upon receiving an incoming message from another endpoint in the group for the first time (see Section 6.2 and Section 6.4). Each Recipient Context matches the Sender Context of the endpoint from which protected messages are received. Besides, in addition to what is defined in [RFC8613], each Recipient Context stores also the public key of the associated other endpoint from which messages are received. Note that each Recipient Context includes a Replay Window, unless the recipient acts only as client and hence processes only responses as incoming messages.

The table in Figure 1 overviews the new information included in the OSCORE Security Context, with respect to what defined in Section 3 of [RFC8613].

Context portion	New information
Common Context	Counter signature algorithm
Common Context	Counter signature parameters
Sender Context	Endpoint's own private key
Each Recipient Context	Public key of the associated other endpoint

Figure 1: Additions to the OSCORE Security Context

Upon receiving a secure CoAP message, a recipient uses the sender's public key, in order to verify the counter signature of the COSE Object (see Section 3).

If not already stored in the Recipient Context associated to the sender, the recipient retrieves the sender's public key from the Group Manager, which collects public keys upon endpoints' joining the group, acts as trusted key repository and ensures the correct association between the public key and the identifier of the sender, for instance by means of public key certificates.

For very constrained devices, it may be not feasible to simultaneously handle the ongoing processing of a just received message and the parallel retrieval of the sender's public key. Such devices can be configured to drop that received message altogether, switch to the retrieval of the sender's public key, and thus have it available to verify following messages from that sender.

Note that a group member can retrieve public keys from the Group Manager and generate the Recipient Context associated to another group member at any point in time, as long as this is done before verifying a received secure CoAP message. The exact configuration is application dependent. For example, an application can configure a group member to retrieve all the required information and to create the Recipient Context exactly upon receiving a message from another group member for the first time. As an alternative, the application can configure a group member to asynchronously retrieve the required information and update its list of Recipient Contexts well before receiving any message, e.g. by Observing [RFC7641] the Group Manager to get updates on the group membership.

It is RECOMMENDED that the Group Manager collects public keys and provides them to group members upon request as described in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz]. Further details about how public keys can be handled and retrieved in the group is out of the scope of this document.

An endpoint receives its own Sender ID from the Group Manager upon joining the group. That Sender ID is valid only within that group, and is unique within the group. An endpoint uses its own Sender ID (together with other data) to generate unique AEAD nonces for outgoing messages, as in [RFC8613]. Endpoints which are configured only as silent servers do not have a Sender ID.

The Sender/Recipient Keys and the Common IV are derived according to the same scheme defined in Section 3.2 of [RFC8613]. The mandatory-to-implement HKDF and AEAD algorithms for Group OSCORE are the same as in [RFC8613].

2.1. Management of Group Keying Material

In order to establish a new Security Context in a group, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter **MUST** be distributed. When doing so, a new value for the Master Salt parameter **MAY** also be distributed, and the Group Manager **SHOULD** preserve the current value of the Sender ID of each group member. An example of Gid format supporting this operation is provided in Appendix C. Then, each group member re-derives the keying material stored in its own Sender Context and Recipient Contexts as described in Section 2, using the updated Gid.

After a new Gid has been distributed, a same Recipient ID ('kid') should not be considered as a persistent and reliable indicator of the same group member. Such an indication can be actually achieved only by verifying countersignatures of received messages.

As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages. Applications may define policies to delete (long-)unused Recipient Contexts and reduce the impact on storage space.

If the application requires so (see Appendix A.1), it is **RECOMMENDED** to adopt a group key management scheme, and securely distribute a new value for the Gid and for the Master Secret parameter of the group's Security Context, before a new joining endpoint is added to the group or after a currently present endpoint leaves the group. This is necessary to preserve backward security and forward security in the group, if the application requires it.

The specific approach used to distribute the new Gid and Master Secret parameter to the group is out of the scope of this document. However, it is **RECOMMENDED** that the Group Manager supports the distribution of the new Gid and Master Secret parameter to the group according to the Group Rekeying Process described in [I-D.ietf-ace-key-groupcomm-oscore].

2.2. Wrap-Around of Partial IVs

An endpoint can eventually experience a wrap-around of its own Sender Sequence Number, which is incremented after sending each new message including a Partial IV. This is the case for all group requests, all Observe notifications [RFC7641] and, optionally, any other response.

When a wrap-around happens, the endpoint **MUST NOT** transmit further messages including a Partial IV until it has derived a new Sender Context, in order to avoid reusing nonces with the same keys.

Furthermore, the endpoint SHOULD inform the Group Manager, that can take one of the following actions:

- o The Group Manager renews the OSCORE Security Context in the group (see Section 2.1).
- o The Group Manager provides a new Sender ID value to the endpoint that has experienced the wrap-around. Then, the endpoint derives a new Sender Context using the new Sender ID, as described in Section 3.2 of [RFC8613].

Either case, same considerations from Section 2.1 hold about possible retaining of stale Recipient Contexts.

3. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. For Group OSCORE, the following modifications apply.

3.1. Updated external_aad

The external_aad of the Additional Authenticated Data (AAD) is extended as follows. In particular, it has one structure used for the encryption process producing the ciphertext, and one structure used for the signing process producing the counter signature.

3.1.1. Updated external_aad for Encryption

The first external_aad structure used for the encryption process producing the ciphertext (see Section 5.3 of [RFC8152]) includes also the counter signature algorithm and related parameters used to sign messages. In particular, compared with Section 5.4 of [RFC8613], the 'algorithms' array in the aad_array MUST also include:

- o 'alg_countersign', which contains the Counter Signature Algorithm from the Common Context (see Section 2). This parameter has the value specified in the "Value" field of the Counter Signature Parameters Registry (see Section 9.1) for this counter signature algorithm.

The 'algorithms' array in the aad_array MAY also include:

- o 'par_countersign', which contains the Counter Signature Parameters from the Common Context (see Section 2). This parameter contains the counter signature parameters encoded as specified in the

"Parameters" field of the Counter Signature Parameters Registry (see Section 9.1), for the used counter signature algorithm. If the Counter Signature Parameters in the Common Context is empty, 'par_countersign' MUST be encoding the CBOR simple value Null.

- o 'par_countersign_key', which contains the Counter Signature Key Parameters from the Common Context (see Section 2). This parameter contains the counter signature key parameters encoded as specified in the "Parameters" field of the Counter Signature Key Parameters Registry (see Section 9.2), for the used counter signature algorithm. If the Counter Signature Key Parameters in the Common Context is empty, 'par_countersign_key' MUST be encoding the CBOR simple value Null.

Thus, the following external_aad structure is used for the encryption process producing the ciphertext (see Section 5.3 of [RFC8152]).

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr,
                alg_countersign : int / tstr,
                par_countersign : any / nil,
                par_countersign_key : any / nil],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr
]
```

3.1.2. Updated external_aad for Signing

The second external_aad structure used for the signing process producing the counter signature as defined below includes also:

- o the counter signature algorithm and related parameters used to sign messages, encoded as in the external_aad structure defined in Section 3.1.1;
- o the value of the OSCORE Option included in the OSCORE message, encoded as a binary string.

Thus, the following external_aad structure is used for the signing process producing the counter signature, as defined below.

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr,
                alg_countersign : int / tstr,
                par_countersign : any / nil,
                par_countersign_key : any / nil],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
  OSCORE_option: bstr
]
```

Note for implementation: this requires the value of the OSCORE option to be fully ready, before starting the signing process.

3.2. Use of the 'kid' Parameter

The value of the 'kid' parameter in the 'unprotected' field of response messages MUST be set to the Sender ID of the endpoint transmitting the message. That is, unlike in [RFC8613], the 'kid' parameter is always present in all messages, i.e. both requests and responses.

3.3. Updated 'unprotected' Field

The 'unprotected' field MUST additionally include the following parameter:

- o CounterSignature0 : its value is set to the counter signature of the COSE object, computed by the sender using its own private key as described in Appendix A.2 of [RFC8152]. In particular, the Sig_structure contains the external_aad as defined in Section 3.1.2 and the ciphertext of the COSE_Encrypt0 object as payload.

4. OSCORE Header Compression

The OSCORE compression defined in Section 6 of [RFC8613] is used, with the following additions for the encoding of the OSCORE Option and the OSCORE Payload.

4.1. Encoding of the OSCORE Option Value

Analogously to [RFC8613], the value of the OSCORE option SHALL contain the OSCORE flag bits, the Partial IV parameter, the kid

context parameter (length and value), and the kid parameter, with the following modifications:

- o The first byte, containing the OSCORE flag bits, has the following encoding modifications:
 - * The fourth least significant bit MUST be set to 1 in every message, to indicate the presence of the 'kid' parameter for all group requests and responses. That is, unlike in [RFC8613], the 'kid' parameter is always present in all messages.
 - * The fifth least significant bit MUST be set to 1 for group requests, to indicate the presence of the 'kid context' parameter in the compressed COSE object. The 'kid context' MAY be present in responses if the application requires it. In such a case, the kid context flag MUST be set to 1.

The flag bits are registered in the OSCORE Flag Bits registry specified in Section 13.7 of [RFC8613].

- o The 'kid context' value encodes the Group Identifier value (Gid) of the group's Security Context.
- o The remaining bytes in the OSCORE Option value encode the value of the 'kid' parameter, which is always present both in group requests and in responses.

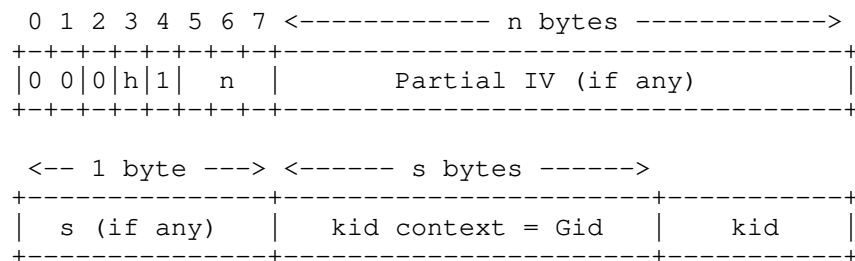


Figure 2: OSCORE Option Value

4.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object concatenated with the value of the CounterSignature0 of the COSE object, computed as in Appendix A.2 of [RFC8152] according to the Counter Signature Algorithm and Counter Signature Parameters in the Security Context.

4.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for group requests and responses. The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full Security Context, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 4 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the label for the new kid context defined in [RFC8613] has value 10. COUNTERSIGN is the CounterSignature0 byte string as described in Section 3 and is 64 bytes long.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c

Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 9:COUNTERSIGN },
  h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (85 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 03 44 61 6c 25 (7 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

1. Response with ciphertext = 60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

Before compression (88 bytes):

```
[
  h'',
  { 4:h'52', 9:COUNTERSIGN },
  h'60b035059d9ef5667c5a0710823b'
]
```

After compression (80 bytes):

Flag byte: 0b00001000 = 0x08

Option Value: 08 52 (2 bytes)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

5. Message Binding, Sequence Numbers, Freshness and Replay Protection

The requirements and properties described in Section 7 of [RFC8613] also apply to OSCORE used in group communication. In particular, group OSCORE provides message binding of responses to requests, which provides relative freshness of responses, and replay protection of requests.

5.1. Synchronization of Sender Sequence Numbers

Upon joining the group, new servers are not aware of the Sender Sequence Number values currently used by different clients to transmit group requests. This means that, when such servers receive a secure group request from a given client for the first time, they are not able to verify if that request is fresh and has not been replayed or (purposely) delayed. The same holds when a server loses synchronization with Sender Sequence Numbers of clients, for instance after a device reboot.

The exact way to address this issue is application specific, and depends on the particular use case and its synchronization requirements. The list of methods to handle synchronization of Sender Sequence Numbers is part of the group communication policy, and different servers can use different methods.

Appendix E describes three possible approaches that can be considered for synchronization of sequence numbers.

6. Message Processing

Each request message and response message is protected and processed as specified in [RFC8613], with the modifications described in the following sections. The following security objectives are fulfilled, as further discussed in Appendix A.2: data replay protection, group-level data confidentiality, source authentication and message integrity.

As per [RFC7252][RFC7390][I-D.dijk-core-groupcomm-bis], group requests sent over multicast MUST be Non-Confirmable. Thus, senders

should store their outgoing messages for an amount of time defined by the application and sufficient to correctly handle possible retransmissions. However, this does not prevent the acknowledgment of Confirmable group requests in non-multicast environments. Besides, according to Section 5.2.3 of [RFC7252], responses to Non-Confirmable group requests SHOULD be also Non-Confirmable. However, endpoints MUST be prepared to receive Confirmable responses in reply to a Non-Confirmable group request.

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. However, a recipient MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 3, or which is not cryptographically validated in a successful way. Either case, it is RECOMMENDED that the recipient does not send back any error message. This prevents servers from replying with multiple error messages to a client sending a group request, so avoiding the risk of flooding and possibly congesting the group.

6.1. Protecting the Request

A client transmits a secure group request as described in Section 8.1 of [RFC8613], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3 of this specification.
- o In step 4, the encryption of the COSE object is modified as described in Section 3 of this specification. The encoding of the compressed COSE object is modified as described in Section 4 of this specification.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2 of this specification. In particular, the payload of the OSCORE message includes also the counter signature.

6.2. Verifying the Request

Upon receiving a secure group request, a server proceeds as described in Section 8.2 of [RFC8613], with the following modifications.

- o In step 2, the decoding of the compressed COSE object follows Section 4 of this specification. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the server MAY create a new Recipient Context and initializes it according to Section 3 of

[RFC8613], also retrieving the client's public key. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server SHALL stop processing the request.

- o In step 4, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3 of this specification.
- o In step 6, the server also verifies the counter signature using the public key of the client from the associated Recipient Context. If the signature verification fails, the server MAY reply with a 4.00 (Bad Request) response.
- o Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, would prevent attackers from overloading the server's storage and creating processing overhead on the server.

6.3. Protecting the Response

A server that has received a secure group request may reply with a secure response, which is protected as described in Section 8.3 of [RFC8613], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3 of this specification.
- o In step 4, the encryption of the COSE object is modified as described in Section 3 of this specification. The encoding of the compressed COSE object is modified as described in Section 4 of this specification.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2 of this specification. In particular, the payload of the OSCORE message includes also the counter signature.

6.4. Verifying the Response

Upon receiving a secure response message, the client proceeds as described in Section 8.4 of [RFC8613], with the following modifications.

- o In step 2, the decoding of the compressed COSE object is modified as described in Section 3 of this specification. If the received Recipient ID ('kid') does not match with any Recipient Context for

the retrieved Gid ('kid context'), then the client MAY create a new Recipient Context and initializes it according to Section 3 of [RFC8613], also retrieving the server's public key. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.

- o In step 3, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3 of this specification.
- o In step 5, the client also verifies the counter signature using the public key of the server from the associated Recipient Context.
- o Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, would prevent attackers from overloading the client's storage and creating processing overhead on the client.

7. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, as well as ensuring uniqueness of Gids within the set of its OSCORE groups.
2. Defining policies for authorizing the joining of its OSCORE groups.
3. Handling the join process to add new endpoints as group members.
4. Establishing the Common Context part of the Security Context, and providing it to authorized group members during the join process, together with the corresponding Sender Context.
5. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process. This includes ensuring uniqueness of Sender IDs within each of its OSCORE groups.
6. Defining a communication policy for each of its OSCORE groups, and signalling it to new endpoints during the join process.
7. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).

8. Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistent with the key management scheme used in the group joined by the new endpoint.
9. Updating the Gid of its OSCORE groups, upon renewing the respective Security Context.
10. Acting as key repository, in order to handle the public keys of the members of its OSCORE groups, and providing such public keys to other members of the same group upon request. The actual storage of public keys may be entrusted to a separate secure storage device.
11. Validating that the format and parameters of public keys of group members are consistent with the countersignature algorithm and related parameters used in the respective OSCORE group.

8. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE. In addition, source authentication of messages is explicitly ensured by means of counter signatures, as further discussed in Section 8.1.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [RFC8613] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [RFC8613] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [RFC8613] hold for Group OSCORE. This is further discussed in Section 8.2.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [RFC8613] hold for Group OSCORE, with the following difference. The countersignature included in a Group OSCORE message is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 8.6.

As discussed in Section 6.2.3 of [I-D.dijk-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when mounted over IP multicast.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then it goes through aspects

covered in the security considerations of OSCORE (Section 12 of [RFC8613]), and discusses how they hold when Group OSCORE is used.

8.1. Group-level Security

The approach described in this document relies on commonly shared group keying material to protect communication within a group. This has the following implications.

- o Messages are encrypted at a group level (group-level data confidentiality), i.e. they can be decrypted by any member of the group, but not by an external adversary or other external entities.
- o The AEAD algorithm provides only group authentication, i.e. it ensures that a message sent to a group has been sent by a member of that group, but not by the alleged sender. This is why source authentication of messages sent to a group is ensured through a counter signature, which is computed by the sender using its own private key and then appended to the message payload.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of messages in the group to unauthorized entities. However, in many use cases, the devices in the group belong to a common authority and are configured by a commissioner (see Appendix B), which results in a practically limited risk and enables a prompt detection/reaction in case of misbehaving.

8.2. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

- o Uniqueness of Sender IDs within the group is enforced by the Group Manager.
- o The case A in Appendix D.4 of [RFC8613] concerns all group requests and responses including a Partial IV (e.g. Observe notifications). In this case, same considerations from [RFC8613] apply here as well.
- o The case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g. single response to a group request). In this case, same considerations from [RFC8613] apply here as well.

As a consequence, each message encrypted/decrypted with the same Sender Key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

8.3. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material should be adopted.

Especially in dynamic, large-scale, groups where endpoints can join and leave at any time, it is important that the considered group key management scheme is efficient and highly scalable with the group size, in order to limit the impact on performance due to the Security Context and keying material update.

8.4. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed, and new security parameters and keying material have been distributed by the Group Manager. In the following two cases, this may result in misaligned Security Contexts between the sender and the recipient.

In the first case, the sender protects a message using the old Security Context, i.e. before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context, hence not being able to correctly process it. A possible way to ameliorate this issue is to preserve the old, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as second attempt. This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large scale groups. On the other hand, a former (compromised) group member can abusively take advantage of this, and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the retention of old Security Contexts.

In the second case, the sender protects a message using the new Security Context, but the recipient receives that request before having installed the new Security Context. Therefore, the recipient

would not be able to correctly process the request and hence discards it. If the recipient receives the new Security Context shortly after that and the sender endpoint uses CoAP retransmissions, the former will still be able to receive and correctly process the message. In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

8.5. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide.

However, this does not impair the security of the AEAD algorithm. In fact, as long as the Master Secret is different for different groups and this condition holds over time, AEAD keys are different among different groups.

8.6. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same signature key in multiple OSCORE groups, possibly administered by different Group Managers. Also, the same endpoint can register several times in the same group, getting multiple unique Sender IDs. This requires that, when a sender endpoint sends a message to an OSCORE group using a Sender ID, the countersignature included in the message is explicitly bound also to that group and to the used Sender ID.

To this end, the countersignature of each message protected with Group OSCORE is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process (see Section 3.1.2). That is, the countersignature is computed also over: the ID Context (Group ID) and the Partial IV, which are always present in group requests; as well as the Sender ID of the message originator, which is always present in all group requests and responses.

Since the signing process takes as input also the ciphertext of the COSE_Encrypt0 object, the countersignature is bound not only to the intended OSCORE group, hence to the triplet (Master Secret, Master Salt, ID Context), but also to a specific Sender ID in that group and to its specific symmetric key used for AEAD encryption, hence to the quartet (Master Secret, Master Salt, ID Context, Sender ID).

This makes it practically infeasible to perform the attack described below, where a malicious group member injects forged messages to a

different OSCORE group than the originally intended one. Let us consider:

- o Two OSCORE groups G1 and G2, with ID Context (Group ID) Gid1 and Gid2, respectively. Both G1 and G2 use the AEAD cipher AES-CCM-16-64-128, i.e. the MAC of the ciphertext is 8 bytes in size.
- o A victim endpoint V which is member of both G1 and G2, and uses the same signature key in both groups. The endpoint V has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same public key of V in G1 and G2, respectively.
- o A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the symmetric keys associated to V in G1 and G2.

If countersignatures were not computed also over the value of the OSCORE option as discussed above, Z can intercept a group message M1 sent by V to G1, and forge a valid signed message M2 to be injected in G2, making it appear as sent by V and valid to be accepted.

More in detail, Z first intercepts a message M1 sent by V in G1, and tries to forge a message M2, by changing the value of the OSCORE option from M1 as follows: the 'kid context' is changed from G1 to G2; and the 'kid' is changed from Sid1 to Sid2.

If M2 is used as a request message, there is a probability equal to 2^{-64} that the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to V in G2. In such a case, the same unchanged signature would be also valid. Note that Z can check offline if a performed forgery is actually valid before sending the forged message to G2. That is, this attack has a complexity of 2^{64} offline calculations.

If M2 is used as a response, Z can also change the response Partial IV, until the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to V in G2. In such a case, the same unchanged signature would be also valid. Since the Partial IV is 5 bytes in size, this requires 2^{40} operations to test all the Partial IVs, which can be done in real-time. Also, the probability that a single given message M1 can be used to forge a response M2 for a given request is equal to 2^{-24} , since there are more MAC values (8 bytes in size) than Partial IV values (5 bytes in size).

Note that, by changing the Partial IV as discussed above, any member of G1 would also be able to forge a valid signed response message M2 to be injected in G1.

8.7. End-to-end Protection

The same considerations from Section 12.1 of [RFC8613] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. Instead, it is not possible to combine DTLS and Group OSCORE for protecting message exchanges where messages are (also) sent over multicast.

8.8. Security Context Establishment

The use of COSE_Encrypt0 and AEAD to protect messages as specified in this document requires an endpoint to be a member of an OSCORE group.

That is, upon joining the group, the endpoint securely receives from the Group Manager the necessary input parameters, which are used to derive the Common Context and the Sender Context (see Section 2). The Group Manager ensures uniqueness of Sender IDs in the same group.

Each different Recipient Context for decrypting messages from a particular sender can be derived at runtime, at the latest upon receiving a message from that sender for the first time.

Countersignatures of group messages are verified by means of the public key of the respective sender endpoint. Upon nodes' joining, the Group Manager collects such public keys and MUST verify proof-of-possession of the respective private key. Later on, a group member can request from the Group Manager the public keys of other group members.

The joining process can occur, for instance, as defined in [I-D.ietf-ace-key-groupcomm-oscore].

8.9. Master Secret

Group OSCORE derives the Security Context using the same construction as OSCORE, and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the Security Context parameters discussed in Section 3.3 of [RFC8613] hold for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by

generating and distributing a new Master Secret. Randomness requirements for security are described in [RFC4086].

8.10. Replay Protection

As in OSCORE, also Group OSCORE relies on sender sequence numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

As discussed in Section 5.1, an endpoint that has just joined a group is exposed to replay attack, as it is not aware of the sender sequence numbers currently used by other group members. Appendix E describes how endpoints can synchronize with senders' sequence numbers.

Unless exchanges in a group rely only on unicast messages, Group OSCORE cannot be used with reliable transport. Thus, unless only unicast messages are sent in the group, it cannot be defined that only messages with sequence numbers that are equal to the previous sequence number + 1 are accepted.

The processing of response messages described in Section 6.4 also ensures that a client accepts a single valid response to a given request from each replying server, unless CoAP observation is used.

8.11. Client Aliveness

As discussed in Section 12.5 of [RFC8613], a server may use the Echo option [I-D.ietf-core-echo-request-tag] to verify the aliveness of the client that originated a received request. This would also allow the server to (re-)synchronize with the client's sequence number, as well as to ensure that the request is fresh and has not been replayed or (purposely) delayed, if it is the first one received from that client after having joined the group or rebooted (see Appendix E.3).

8.12. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.2, an endpoint that experiences a wrap-around of its own Sender Sequence Number MUST NOT transmit further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonces with the same Sender key.

In order to renew its own Sender Context, the endpoint SHOULD inform the Group Manager, which can either renew the whole Security Context

by means of group rekeying, or provide only that endpoint with a new Sender ID value. Either case, the endpoint derives a new Sender Context, and in particular a new Sender Key.

Additionally, the same considerations from Section 12.6 of [RFC8613] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

8.13. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

8.14. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [RFC8613] hold for Group OSCORE.

Furthermore, the following privacy considerations hold, about the OSCORE option that may reveal information on the communicating endpoints.

- o The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. Since both requests and responses always include the 'kid' parameter, this may reveal information about both a client sending a group request and all the possibly replying servers sending their own individual response.
- o The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current Group ID is indicated in the 'kid context' parameter. Moreover, this parameter explicitly relates two or more communicating endpoints, as members of the same OSCORE group.

Also, using the mechanisms described in Appendix E.3 to achieve sequence number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the replay window of each known client on a clean shutdown.

9. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[This Document]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

9.1. Counter Signature Parameters Registry

This specification establishes the IANA "Counter Signature Parameters" Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 9.3.

This registry specifies the parameters of each admitted countersignature algorithm, as well as the possible structure they are organized into. This information is used to populate the parameter Counter Signature Parameters of the Common Context (see Section 2).

The columns of this table are:

- o Name: A value that can be used to identify an algorithm in documents for easier comprehension. Its value is taken from the 'Name' column of the "COSE Algorithms" Registry.
- o Value: The value to be used to identify this algorithm. Its content is taken from the 'Value' column of the "COSE Algorithms" Registry. The value MUST be the same one used in the "COSE Algorithms" Registry for the entry with the same 'Name' field.
- o Parameters: This indicates the CBOR encoding of the parameters (if any) for the counter signature algorithm indicated by the 'Value' field.
- o Description: A short description of the parameters encoded in the 'Parameters' field (if any).
- o Reference: This contains a pointer to the public specification for the field, if one exists.

Initial entries in the registry are as follows.

Name	Value	Parameters	Description	Reference
EdDSA	-8	crv : int	crv value taken from the COSE Elliptic Curve Registry	[This Document]
ES256	-7	crv : int	crv value taken from the COSE Elliptic Curve Registry	[This Document]
ES384	-35	crv : int	crv value taken from the COSE Elliptic Curve Registry	[This Document]
ES512	-36	crv : int	crv value taken from the COSE Elliptic Curve Registry	[This Document]
PS256	-37		Parameters not present	[This Document]
PS384	-38		Parameters not present	[This Document]
PS512	-39		Parameters not present	[This Document]

9.2. Counter Signature Key Parameters Registry

This specification establishes the IANA "Counter Signature Key Parameters" Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 9.3.

This registry specifies the parameters of countersignature keys for each admitted countersignature algorithm, as well as the possible structure they are organized into. This information is used to populate the parameter Counter Signature Key Parameters of the Common Context (see Section 2).

The columns of this table are:

- o Name: A value that can be used to identify an algorithm in documents for easier comprehension. Its value is taken from the 'Name' column of the "COSE Algorithms" Registry.
- o Value: The value to be used to identify this algorithm. Its content is taken from the 'Value' column of the "COSE Algorithms" Registry. The value MUST be the same one used in the "COSE Algorithms" Registry for the entry with the same 'Name' field.
- o Parameters: This indicates the CBOR encoding of the key parameters (if any) for the counter signature algorithm indicated by the 'Value' field.
- o Description: A short description of the parameters encoded in the 'Parameters' field (if any).
- o Reference: This contains a pointer to the public specification for the field, if one exists.

Initial entries in the registry are as follows.

Name	Value	Parameters	Description	Reference
EdDSA	-8	[kty : int , crv : int]	kty value is 1, as Key Type "OKP" from the COSE Key Types Registry crv value taken from the COSE	[This Document]

			Elliptic Curve Registry	
ES256	-7	[kty : int , crv : int]	kty value is 2, as Key Type "EC2" from the COSE Key Types Registry crv value taken from the COSE Elliptic Curve Registry	[This Document]
ES384	-35	[kty : int , crv : int]	kty value is 2, as Key Type "EC2" from the COSE Key Types Registry crv value taken from the COSE Elliptic Curve Registry	[This Document]
ES512	-36	[kty : int , crv : int]	kty value is 2, as Key Type "EC2" from the COSE Key Types Registry crv value taken from the COSE Elliptic Curve Registry	[This Document]
PS256	-37	kty : int	kty value is 3, as Key Type "RSA" from the COSE Key Types Registry	[This Document]

PS384	-38	kty : int	kty value is 3, as Key Type "RSA" from the COSE Key Types Registry	[This Document]
PS512	-39	kty : int	kty value is 3, as Key Type "RSA" from the COSE Key Types Registry	[This Document]

9.3. Expert Review Instructions

The IANA Registries established in this document are defined as "Expert Review". This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Experts should inspect the entry for the algorithm considered, to verify the conformity of the encoding proposed against the theoretical algorithm, including completeness of the 'Parameters' column. Expert needs to make sure values are taken from the right registry, when that's required. Expert should consider requesting an opinion on the correctness of registered parameters from the CBOR Object Signing and Encryption Working Group (COSE). Encodings that do not meet these objective of clarity and completeness should not be registered.
- o Duplicated registration and point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments.
- o Experts should take into account the expected usage of fields when approving point assignment. The length of the 'Parameters' encoding should be weighed against the usage of the entry, considering the size of device it will be used on. Additionally, the length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be

used on, and the number of code points left that encode to that size.

- o Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

10. References

10.1. Normative References

- [I-D.dijk-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-01 (work in progress), July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <<https://www.rfc-editor.org/info/rfc6979>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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/info/rfc8613>>.

10.2. Informative References

- [I-D.ietf-ace-key-groupcomm-oscore] Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-03 (work in progress), November 2019.
- [I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-25 (work in progress), October 2019.
- [I-D.ietf-core-echo-request-tag] Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-08 (work in progress), November 2019.
- [I-D.somaraju-ace-multicast] Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-02 (work in progress), October 2016.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

- [RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <<https://www.rfc-editor.org/info/rfc6282>>.
- [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/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document.

A.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- o Multicast communication topology: this document considers both 1-to-N (one sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical Low-power and Lossy Network (LLN). Examples of use cases that benefit from secure group communication are provided in Appendix B.

In a 1-to-N communication model, only a single client transmits data to the group, in the form of request messages; in an M-to-N communication model (where M and N do not necessarily have the same value), M group members are clients. According to [RFC7390], any possible proxy entity is supposed to know about the clients in the group and to not perform aggregation of response messages from

multiple servers. Also, every client expects and is able to handle multiple response messages associated to a same request sent to the group.

- o Group size: security solutions for group communication should be able to adequately support different and possibly large groups. The group size is the current number of members in a group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 clients would be able to properly cover the group sizes required for most use cases that are relevant for this document. The maximum group size is expected to be in the range of 2 to 100 devices. Groups larger than that should be divided into smaller independent groups.
- o Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as group member.
- o Provisioning and management of Security Contexts: an OSCORE Security Context must be established among the group members. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, multicast security policies and security parameters in the group. The actual provisioning and management of the Security Context is out of the scope of this document.
- o Multicast data security ciphersuite: all group members must agree on a ciphersuite to provide authenticity, integrity and confidentiality of messages in the group. The ciphersuite is specified as part of the Security Context.
- o Backward security: a new device joining the group should not have access to any old Security Contexts used before its joining. This ensures that a new group member is not able to decrypt confidential data sent before it has joined the group. The adopted key management scheme should ensure that the Security Context is updated to ensure backward confidentiality. The actual mechanism to update the Security Context and renew the group keying material upon a group member's joining has to be defined as part of the group key management scheme.
- o Forward security: entities that leave the group should not have access to any future Security Contexts or message exchanged within the group after their leaving. This ensures that a former group member is not able to decrypt confidential data sent within the

group anymore. Also, it ensures that a former member is not able to send encrypted and/or integrity protected messages to the group anymore. The actual mechanism to update the Security Context and renew the group keying material upon a group member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The approach described in this document aims at fulfilling the following security objectives:

- o Data replay protection: replayed group request messages or response messages must be detected.
- o Group-level data confidentiality: messages sent within the group shall be encrypted if privacy sensitive data is exchanged within the group. This document considers group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that they can be decrypted by any member of the group, but not by an external adversary or other external entities.
- o Source authentication: messages sent within the group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the group in the first place, and in particular by a specific member of the group.
- o Message integrity: messages sent within the group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with by an external adversary or other external entities which are not group members.
- o Message ordering: it must be possible to determine the ordering of messages coming from a single sender. In accordance with OSCORE [RFC8613], this results in providing relative freshness of group requests and absolute freshness of responses. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [RFC7390][I-D.dijk-core-groupcomm-bis] provides the necessary background for multicast-based CoAP communication, with particular reference to low-power and lossy networks (LLNs) and resource constrained environments. The interested reader is encouraged to first read [RFC7390][I-D.dijk-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that

benefit from secure group communication. Specific security requirements for these use cases are discussed in Appendix A.

- o Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The devices are organized into groups according to their physical location in the building. For instance, lighting devices and switches in a room or corridor can be configured as members of a single group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in a group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical groups to be formed even if devices in the lighting group may be physically in different subnets (e.g. on wired and wireless networks). Connectivity between lighting devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944][RFC6282]. Group communication enables synchronous operation of a group of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large group of luminaires are changed at the same perceived time. This is especially useful for providing a visual synchronicity of light effects to the user. As a practical guideline, events within a 200 ms interval are perceived as simultaneous by humans, which is necessary to ensure in many setups. Devices may reply back to the switches that issue on/off/dimming commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status. In a typical lighting control scenario, a single switch is the only entity responsible for sending commands to a group of lighting devices. In more advanced lighting control use cases, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a group of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.
- o Integrated building control: enabling Building Automation and Control Systems (BACSS) to control multiple heating, ventilation and air-conditioning units to pre-defined presets. Controlled units can be organized into groups in order to reflect their physical position in the building, e.g. devices in the same room can be configured as members of a single group. As a practical guideline, events within intervals of seconds are typically acceptable. Controlled units are expected to possibly reply back to the BACS issuing control commands, in order to report about the

execution of the requested operation (e.g. OK, failure, error) and their current operational status.

- o Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a Low-power and Lossy Network (LLN) that is otherwise typically used to deal with only small amounts of data, on an infrequent base. Rather than sending software and firmware updates as unicast messages to each individual device, multicasting such updated data to a larger group of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates are expected to possibly reply back, in order to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.
- o Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a group of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.
- o Commissioning of Low-power and Lossy Network (LLN) systems: a commissioning device is responsible for querying all devices in the local network or a selected subset of them, in order to discover their presence, and be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features, or sharing a common physical location can be configured as members of a single group. Queried devices are expected to reply back to the commissioning device, in order to notify their presence, and provide the requested information and their current operational status.
- o Emergency multicast: a particular emergency related information (e.g. natural disaster) is generated and multicast by an emergency notifier, and relayed to multiple devices. The latter may reply back to the emergency notifier, in order to provide their feedback

and local information related to the ongoing emergency. This kind of setups should additionally rely on a fault tolerance multicast algorithm, such as Multicast Protocol for Low-Power and Lossy Networks (MPL).

Appendix C. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be specifically formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

For each group, the Group Prefix is constant over time and is uniquely defined in the set of all the groups associated to the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. The size of the Group Prefix directly impact on the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 upon completing each renewal of the Security Context and keying material in the group (see Section 2.1). In particular, once a new Master Secret has been distributed to the group, all the group members increment by 1 the Group Epoch in the Group Identifier of that group.

As an example, a 3-byte Group Identifier can be composed of: i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string; and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Common Context 61532 times in the group, its Group Identifier will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group assumes that enough time elapses between two consecutive usages of the same Group Epoch value in that group. This ensures that the Gid value is temporally unique during the lifetime of a given message. Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 8.5, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers, and has to try using different Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favourable that Group Identifiers from different Group Managers have a size that

result in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

Communications between a joining endpoint and the Group Manager rely on the CoAP protocol and must be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join the group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the Security Context (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

It is RECOMMENDED that the join process adopts the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by server endpoints to synchronize with sender sequence numbers of client endpoints sending group requests.

E.1. Best-Effort Synchronization

Upon receiving a group request from a client, a server does not take any action to synchronize with the sender sequence number of that client. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

E.2. Baseline Synchronization

Upon receiving a group request from a given client for the first time, a server initializes its last-seen sender sequence number in its Recipient Context associated to that client. However, the server drops the group request without delivering it to the application layer. This provides a reference point to identify if future group requests from the same client are fresher than the last one received.

A replay time interval exists, between when a possibly replayed or delayed message is originally transmitted by a given client and the first authentic fresh message from that same client is received. This can be acceptable for use cases where servers admit such a trade-off between performance and assurance of message freshness.

E.3. Challenge-Response Synchronization

A server performs a challenge-response exchange with a client, by using the Echo Option for CoAP described in Section 2 of [I-D.ietf-core-echo-request-tag] and according to Appendix B.1.2 of [RFC8613].

That is, upon receiving a group request from a particular client for the first time, the server processes the message as described in Section 6.2 of this specification, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with an OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The server stores the option value included therein.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, a client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. In particular, the client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This makes it possible for the client to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. Either case, the client uses the sender sequence number value currently stored in its own Sender Context. If the client stores group requests for possible retransmission with the Echo Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed as a message, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 3).

Upon receiving the unicast request including the Echo Option, the server verifies that the option value equals the stored and previously sent value; otherwise, the request is silently discarded. Then, the server verifies that the unicast request has been received within a pre-configured time interval, as described in [I-D.ietf-core-echo-request-tag]. In such a case, the request is further processed and verified; otherwise, it is silently discarded. Finally, the server updates the Recipient Context associated to that client, by setting the Replay Window according to the Sequence Number from the unicast request conveying the Echo Option. The server either delivers the request to the application if it is an actual retransmission of the original one, or discards it otherwise. Mechanisms to signal whether the resent request is a full retransmission of the original one are out of the scope of this specification.

In case it does not receive a valid unicast request including the Echo Option within the configured time interval, the server endpoint should perform the same challenge-response upon receiving the next group request from that same client.

A server should not deliver group requests from a given client to the application until one valid request from that same client has been verified as fresh, as conveying an echoed Echo Option [I-D.ietf-core-echo-request-tag]. Also, a server may perform the challenge-response described above at any time, if synchronization with sender sequence numbers of clients is (believed to be) lost, for instance after a device reboot. It is the role of the application to define under what circumstances sender sequence numbers lose synchronization. This can include a minimum gap between the sender sequence number of the latest accepted group request from a client and the sender sequence number of a group request just received from the same client. A client has to be always ready to perform the challenge-response based on the Echo Option in case a server starts it.

Note that endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.01 (Unauthorized) response to the client. Therefore, silent servers should adopt alternative approaches to achieve and maintain synchronization with sender sequence numbers of clients.

This approach provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large groups where many endpoints at the same time might join as new members or lose synchronization.

Appendix F. No Verification of Signatures

There are some application scenarios using group communication that have particularly strict requirements. One example of this is the requirement of low message latency in non-emergency lighting applications [I-D.somaraju-ace-multicast]. For those applications which have tight performance constraints and relaxed security requirements, it can be inconvenient for some endpoints to verify digital signatures in order to assert source authenticity of received messages. In other cases, the signature verification can be deferred or only checked for specific actions. For instance, a command to turn a bulb on where the bulb is already on does not need the signature to be checked. In such situations, the counter signature needs to be included anyway as part of the message, so that an endpoint that needs to validate the signature for any reason has the ability to do so.

In this specification, it is NOT RECOMMENDED that endpoints do not verify the counter signature of received messages. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation is that security in the group is based only on the group-authenticity of the shared keying material used for encryption. That is, endpoints in the group have evidence that a received message has been originated by a group member, although not specifically identifiable in a secure way. This can violate a number of security requirements, as the compromise of any element in the group means that the attacker has the ability to control the entire group. Even worse, the group may not be limited in scope, and hence the same keying material might be used not only for light bulbs but for locks as well. Therefore, extreme care must be taken in situations where the security requirements are relaxed, so that deployment of the system will always be done safely.

Appendix G. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

G.1. Version -05 to -06

- o Group IDs mandated to be unique under the same Group Manager.
- o Clarifications on parameter update upon group rekeying.
- o Updated external_aad structures.
- o Dynamic derivation of Recipient Contexts made optional and application specific.

- o Optional 4.00 response for failed signature verification on the server.
- o Removed client handling of duplicated responses to multicast requests.
- o Additional considerations on public key retrieval and group rekeying.
- o Added Group Manager responsibility on validating public keys.
- o Updates IANA registries.
- o Reference to RFC 8613.
- o Editorial improvements.

G.2. Version -04 to -05

- o Added references to draft-dijk-core-groupcomm-bis.
- o New parameter Counter Signature Key Parameters (Section 2).
- o Clarification about Recipient Contexts (Section 2).
- o Two different external_aad for encrypting and signing (Section 3.1).
- o Updated response verification to handle Observe notifications (Section 6.4).
- o Extended Security Considerations (Section 8).
- o New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

G.3. Version -03 to -04

- o Added the new "Counter Signature Parameters" in the Common Context (see Section 2).
- o Added recommendation on using "deterministic ECDSA" if ECDSA is used as counter signature algorithm (see Section 2).
- o Clarified possible asynchronous retrieval of key material from the Group Manager, in order to process incoming messages (see Section 2).

- o Structured Section 3 into subsections.
- o Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).
- o Clarified non reliability of 'kid' as identity indicator for a group member (see Section 2.1).
- o Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).
- o The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).
- o Updated examples of compressed COSE object, now with the sixth less significant bit in the Flag Byte of the OSCORE option value set to 0 (see Section 4.3).
- o Relaxed statements on sending error messages (see Section 6).
- o Added explicit step on computing the counter signature for outgoing messages (see Sections 6.1 and 6.3).
- o Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).
- o Handling of replied/repeated responses on the client (see Section 6.4).
- o New IANA Registry "Counter Signature Parameters" (see Section 9.1).

G.4. Version -02 to -03

- o Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.
- o Added discussion on wrap-Around of Partial IVs (see Section 2.2).
- o Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).
- o The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).
- o Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".

- o Clarifications about Non-Confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".
- o Clarifications about error handling in Section 6 "Message Processing".
- o Compacted list of responsibilities of the Group Manager in Section 7.
- o Revised and extended security considerations in Section 8.
- o Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.
- o Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

G.5. Version -01 to -02

- o Terminology has been made more aligned with RFC7252 and draft-ietf-core-object-security: i) "client" and "server" replace the old "multicaster" and "listener", respectively; ii) "silent server" replaces the old "pure listener".
- o Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.
- o Section 3 has been updated with the new format of the Additional Authenticated Data.
- o Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.
- o Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.
- o Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.
- o Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.
- o Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.

- o Minor updates to Appendix E.3 about Challenge-Response synchronization of sequence numbers based on the Echo option from draft-ietf-core-echo-request-tag.

G.6. Version -00 to -01

- o Section 1.1 has been updated with the definition of group as "security group".
- o Section 2 has been updated with:
 - * Clarifications on establishment/derivation of Security Contexts.
 - * A table summarizing the the additional context elements compared to OSCORE.
- o Section 3 has been updated with:
 - * Examples of request and response messages.
 - * Use of CounterSignature0 rather than CounterSignature.
 - * Additional Authenticated Data including also the signature algorithm, while not including the Group Identifier any longer.
- o Added Section 6, listing the responsibilities of the Group Manager.
- o Added Appendix A (former section), including assumptions and security objectives.
- o Appendix B has been updated with more details on the use cases.
- o Added Appendix C, providing an example of Group Identifier format.
- o Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, John Mattsson, Dave Robin, Jim Schaad, Ludwig Seitz and Peter van der Stok for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
July 08, 2019

CoRE Resource Directory
draft-ietf-core-resource-directory-23

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

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 January 9, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Architecture and Use Cases	6
3.1. Principles	6
3.2. Architecture	7
3.3. RD Content Model	8
3.4. Link-local addresses and zone identifiers	12
3.5. Use Case: Cellular M2M	12
3.6. Use Case: Home and Building Automation	13
3.7. Use Case: Link Catalogues	14
4. RD discovery and other interface-independent components	14
4.1. Finding a Resource Directory	15
4.1.1. Resource Directory Address Option (RDAO)	17
4.2. Payload Content Formats	18
4.3. URI Discovery	19
5. Registration	21
5.1. Simple Registration	26
5.2. Third-party registration	28
5.3. Operations on the Registration Resource	29
5.3.1. Registration Update	29
5.3.2. Registration Removal	32
5.3.3. Further operations	33
6. RD Lookup	33
6.1. Resource lookup	34
6.2. Lookup filtering	34
6.3. Resource lookup examples	36
6.4. Endpoint lookup	39
7. Security policies	40
7.1. Secure RD discovery	41
7.2. Secure RD filtering	42
7.3. Secure endpoint Name assignment	42

8.	Security Considerations	42
8.1.	Endpoint Identification and Authentication	42
8.2.	Access Control	43
8.3.	Denial of Service Attacks	43
9.	IANA Considerations	44
9.1.	Resource Types	44
9.2.	IPv6 ND Resource Directory Address Option	44
9.3.	RD Parameter Registry	44
9.3.1.	Full description of the "Endpoint Type" Registration Parameter	46
9.4.	"Endpoint Type" (et=) RD Parameter values	46
9.5.	Multicast Address Registration	47
10.	Examples	47
10.1.	Lighting Installation	48
10.1.1.	Installation Characteristics	48
10.1.2.	RD entries	49
10.2.	OMA Lightweight M2M (LWM2M) Example	52
10.2.1.	The LWM2M Object Model	52
10.2.2.	LWM2M Register Endpoint	54
10.2.3.	LWM2M Update Endpoint Registration	55
10.2.4.	LWM2M De-Register Endpoint	56
11.	Acknowledgments	56
12.	Changelog	56
13.	References	66
13.1.	Normative References	66
13.2.	Informative References	66
Appendix A.	Groups Registration and Lookup	68
Appendix B.	Web links and the Resource Directory	70
B.1.	A simple example	70
B.1.1.	Resolving the URIs	71
B.1.2.	Interpreting attributes and relations	71
B.2.	A slightly more complex example	71
B.3.	Enter the Resource Directory	72
B.4.	A note on differences between link-format and Link headers	74
Appendix C.	Limited Link Format	75
Authors' Addresses	75

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying `"/.well-known/core"`. In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

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 [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is `_resolved against_` a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Sector

In the context of a Resource Directory, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the Resource Directory to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the Resource Directory (RD) containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during the installation of the network by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing a Resource Directory's address.

3. Architecture and Use Cases

3.1. Principles

The Resource Directory is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would be limiting those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the resource directory if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the resource directory can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory. Changes to the information in the Resource Directory do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain resource directory registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to resource directory registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

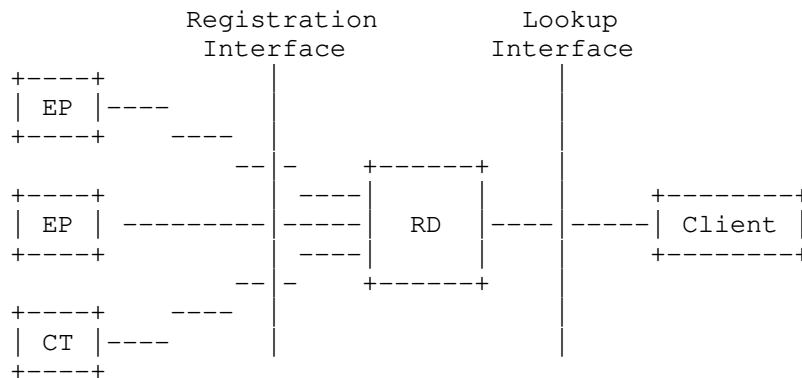


Figure 1: The resource directory architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of /.well-known/core and the resource directory respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and a Resource Directory. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

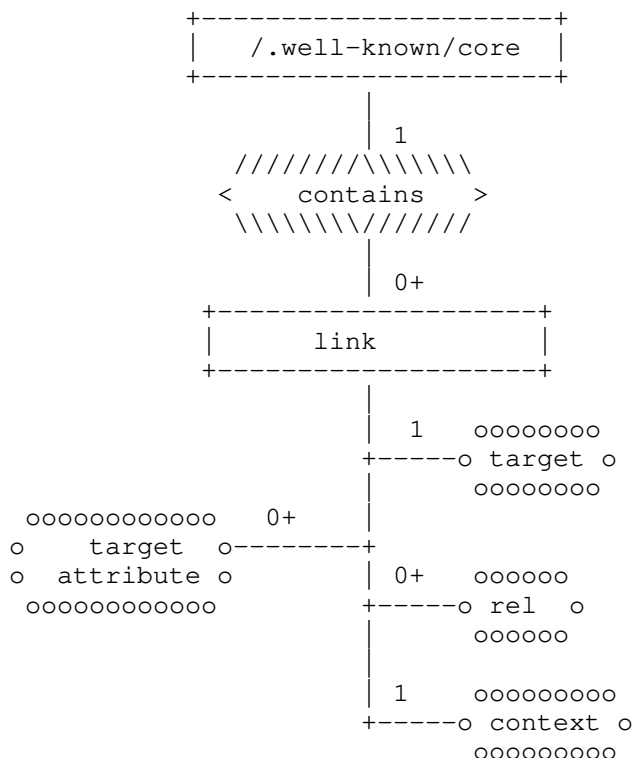


Figure 2: E-R Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- o a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its ".well-known/core". Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- o Zero or more link relations: They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

- o A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute. It defaults to that document's URI.

- o A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- o Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

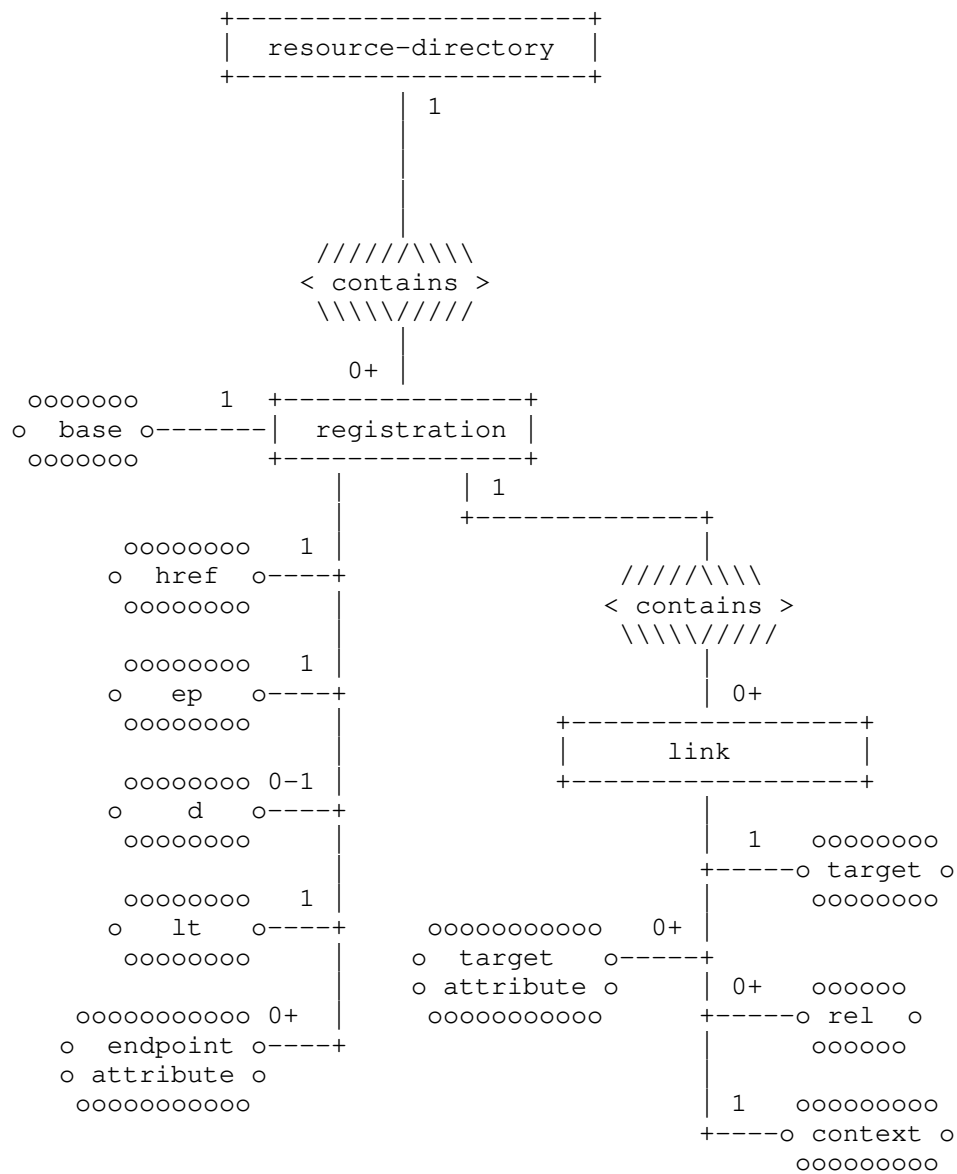


Figure 3: E-R Model of the content of the Resource Directory

The model shown in Figure 3 models the contents of the resource directory which contains in addition to /.well-known/core:

- o 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- o an endpoint name ("ep", a Unicode string) unique within a sector
- o a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- o a lifetime ("lt"),
- o a registration resource location inside the RD ("href"),
- o optionally a sector ("d", a Unicode string)
- o optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. [I-D.silverajan-core-coap-protocol-negotiation]). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those can not be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers can not announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that

can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with a Resource Directory, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border router (6LBR) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no 6LBR is connected, and the network only supports the IP communication between the connected nodes. The installation phase is usually followed by the operational phase.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. Resource Directory can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to a Resource Directory. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by Resource Directories. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between a Resource Directory (RD), endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. Only multicast discovery operations are not possible on HTTP, and Simple Registration can not be executed as base attribute (which is mandatory for HTTP) can not be used there. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs / CTs, lookup clients, RD servers during simple registrations) MUST be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they should retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and fall back to link-format when receiving 4.15 (Unsupported Content Format; 415 in HTTP).

A resource directory MAY make the information submitted to it available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more resource directories for discovery purposes. Dependent on the operational conditions, one or more of the techniques below apply. The use of DNS-SD [RFC6763] is described in [I-D.ietf-core-rd-dns-sd].

The device may be pre-configured to exercise specific mechanisms for finding the resource directory:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be

preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)

2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

For cases where the device is not specifically configured with a way to find a resource directory, the network may want to provide a suitable default.

1. If the address configuration of the network is performed via SLAAC, this is provided by the RDAO option Section 4.1.1.
2. If the address configuration of the network is performed via DHCP, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable resource directory.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as a resource directory (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a Unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local address, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for registration.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the

candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- o In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- o In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- o The use of DNS facilities is described in [I-D.ietf-core-rd-dns-sd].

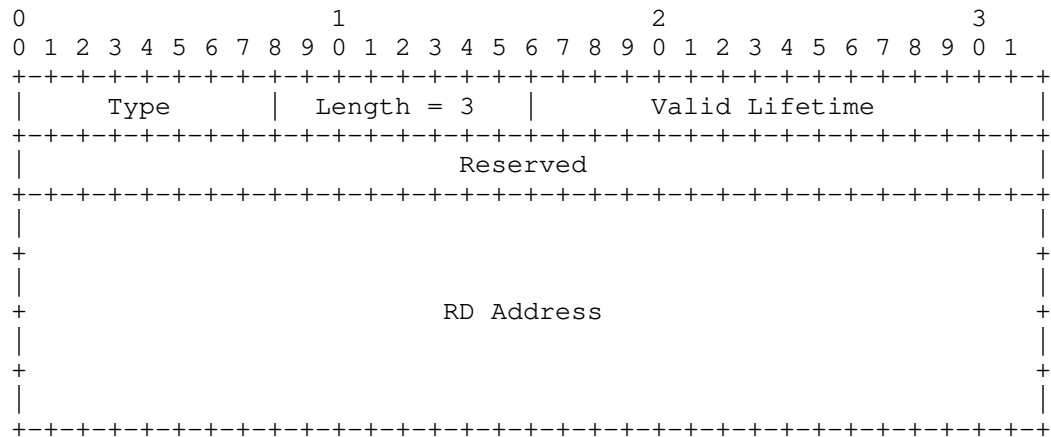
The use of multicast discovery in mesh networks is NOT recommended.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) using IPv6 Neighbor Discovery (ND) carries information about the address of the Resource Directory (RD). This information is needed when endpoints cannot discover the Resource Directory with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many resource directory addresses.

The RDAO format is:



Fields:

Type:	38
Length:	8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.
Valid Lifetime:	16-bit unsigned integer. The length of time in units of 60 seconds (relative to the time the packet is received) that this Resource Directory address is valid. A value of all zero bits (0x0) indicates that this Resource Directory address is not valid anymore.
Reserved:	This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.
RD Address:	IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.2. Payload Content Formats

Resource Directory implementations using this specification MUST support the application/link-format content format (ct=40).

Resource Directories implementing this specification MAY support additional content formats.

Any additional content format supported by a Resource Directory implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690]. A complete set of RD discovery methods is described in Section 4.1.

Discovery of the RD registration URI path is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup*"` is used to discover the URIs for RD Lookup operations, `core.rd*` is used to discover all URI paths for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the `"ct"` target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. The well-known entry points SHOULD be provided to enable unicast discovery.

An implementation of this resource directory specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP and Client -> RD

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type. SHOULD contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", or "core.rd*"

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct=40,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40,

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple

content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*
```

```
Res: 2.05 Content
```

```
</rd>;rt="core.rd";ct="40 65225",  
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504",
```

Figure 6: Example discovery exchange indicating additional content-formats

From a management and maintenance perspective, it is necessary to identify the components that constitute the RD server. The identification refers to information about for example client-server incompatibilities, supported features, required updates and other aspects. The URI discovery address, as described in section 4 of [RFC6690] can be used to find the identification.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info
```

```
Res: 2.05 Content
```

```
<http://software.example.com/shiny-resource-directory/1.0beta1>;  
  rel="impl-info"
```

Figure 7: Example exchange of obtaining implementation information

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root, at the discovery site (as in this example) or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message

payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- o When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- o When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), the RD only needs to accept representations in Limited Link Format as described there. Its behavior with representations outside that subset is implementation defined.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `{+rd}{?ep,d,lt,base,extra-attrs*}`

URI Template Variables:

`rd` := RD registration URI (mandatory). This is the location of the RD, as obtained from discovery.

`ep` := Endpoint name (mostly mandatory). The endpoint name is an identifier that MUST be unique within a sector. As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoding) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159. The maximum length of this parameter is 63 UTF-8 encoded bytes. If the RD is configured to recognize the endpoint (e.g. based on its security context), the RD assigns an endpoint name based on a set of configuration parameter values.

`d` := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector or leave it empty. The sector is encoded like the `ep` parameter, and is limited to 63 UTF-8 encoded bytes as well. The endpoint name and sector name are not set when one or both are set in an accompanying authorization token.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

`base` := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and

":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that can not be reached by potential lookup clients at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component can not meaningfully use [RFC6690] Link Format due to its prevalence of the Origin concept in relative reference resolution. Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional).
The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a "/.well-known/core" response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

A Resource Directory may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req: POST /rd?ep=nodel&base=http://[2001:db8:1::1] HTTP/1.1
Host: example.com
Content-Type: application/link-format
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- o The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- o The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/core"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

Req: POST /.well-known/core?lt=6000&ep=node1
(No payload)

Figure 10: First half example exchange of a simple registration

- o The Resource Directory queries the registrant-ep's discovery resource to determine the success of the operation. It SHOULD keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

Req: GET /.well-known/core
Accept: 40

Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

Res: 2.04 Changed

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the caching item was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to "/.well-known/core" after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /.well-known/core{?ep,d,lt,extra-attrs*}

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one.

The following response is expected on this interface:

Success: 2.04 "Changed".

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the Resource Directory from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. An endpoint SHOULD NOT use this interface for registrations that it did not create. The registrations are resources of the RD.

After the initial registration, the registering endpoint retains the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the Registration Resource is removed, the corresponding endpoint will need to be re-registered.

The Registration Resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

These operations are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update the lifetime or the base URI registration parameters "lt", "base" as in Section 5. Parameters that are not being changed SHOULD NOT be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters MUST be included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration, following the same restrictions as in the registration. If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified. If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of

whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have been removed).

If the registration fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint updates its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- o endpoint name (ep)=endpoint1
- o lifetime (lt)=500
- o Base URI (base)=coap://local-proxy-old.example.com:5683
- o payload of Figure 8

The initial state of the Resource Directory is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://local-proxy-old.example.com:5683/sensors/temp>;ct=41;
  rt="temperature-c";if="sensor";
  anchor="coap://local-proxy-old.example.com:5683/",
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com:5683/sensors/temp";rel="described
by"
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?base=coaps://new.example.com:5684

Res: 2.04 Changed
```

Figure 15: Example registration update that changes the base address

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://new.example.com:5684/sensors/temp>;ct=41;
  rt="temperature-c";if="sensor";
  anchor="coap://new.example.com:5684/",
<http://www.example.com/sensors/temp>;
  anchor="coap://new.example.com:5684/sensors/temp";rel="describedby"
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- o Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- o Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup **MUST** return a list of endpoints and a resource lookup **MUST** return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD. The links and link parameters returned by the lookup are equal to the submitted ones, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the base URI of the registration as the anchor. Links of which href or anchor was submitted as a (full) URI are returned with these attributes unmodified.

Above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The Resource Directory **MAY** replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD **MUST NOT** show its links unless the lookup was made from the same link. The RD **MUST NOT** include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

The page and count parameters are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The Resource Directory MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "link-type" match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=core.s" matches ";if="abc core.s";"). A resource link also matches a search criterion if its endpoint would match the criterion, and viceversa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables:

type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup. The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

```
Req: GET /rd-lookup/res?rt=temperature
```

```
Res: 2.05 Content
```

```
<coap://[2001:db8:3::123]:61616/temp>;rt="temperature";  
    anchor="coap://[2001:db8:3::123]:61616"
```

Figure 18: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=light
```

```
Observe: 0
```

```
Res: 2.05 Content
```

```
Observe: 23
```

```
Payload: empty
```

(at a later point in time)

```
Res: 2.05 Content
```

```
Observe: 24
```

```
Payload:
```

```
<coap://[2001:db8:3::124]/west>;rt="light";  
    anchor="coap://[2001:db8:3::124] ",  
<coap://[2001:db8:3::124]/south>;rt="light";  
    anchor="coap://[2001:db8:3::124] ",  
<coap://[2001:db8:3::124]/east>;rt="light";  
    anchor="coap://[2001:db8:3::124] "
```

Figure 19: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content

```
<coap://[2001:db8:3::123]:61616/res/0>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/1>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/2>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/3>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/4>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616"
```

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content

```
<coap://[2001:db8:3::123]:61616/res/5>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/6>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/7>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/8>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/9>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616"
```

Figure 20: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th request of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```

Req: GET /rd-lookup/res?et=oic.d.sensor

<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index";
  anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/temp>;rt="temperature-c";
  if="sensor"; anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/light>;rt="light-lux";
  if="sensor"; anchor="coap://sensor1.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel="alternate";
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index";
  anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/temp>;rt="temperature-c";
  if="sensor"; anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/light>;rt="light-lux";
  if="sensor"; anchor="coap://sensor2.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
  anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel="alternate";
  anchor="coap://sensor2.example.com/sensors/temp"

```

Figure 21: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns registration resources which can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as absolute references. (This avoids the RFC6690 ambiguities.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

A Resource Directory can report endpoints in lookup that are not hosted at the same address. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint type (et) lookup with the value oic.d.sensor (which is currently a registered rt value):

```
Req: GET /rd-lookup/ep?et=oic.d.sensor
```

```
Res: 2.05 Content
```

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep="node5";  
et="oic.d.sensor";ct="40";rt="core.rd-ep",  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";  
et="oic.d.sensor";ct="40";d="floor-3";rt="core.rd-ep"
```

Figure 22: Examples of endpoint lookup

7. Security policies

The Resource Directory (RD) provides assistance to applications situated on a selection of nodes to discover endpoints on connected nodes. This section discusses different security aspects of accessing the RD.

The contents of the RD are inserted in two ways:

1. The node hosting the discoverable endpoint fills the RD with the contents of /.well-known/core by:
 - * Storing the contents directly into RD (see Section 5)
 - * Requesting the RD to load the contents from /.well-known/core (see Section 5.1)
2. A Commissioning Tool (CT) fills the RD with endpoint information for a set of discoverable nodes. (see Section 5 with base=authority parameter value)

In both cases, the nodes filling the RD should be authenticated and authorized to change the contents of the RD. An Authorization Server (AS) is responsible to assign a token to the registering node to

authorize the node to discover or register endpoints in a given RD [I-D.ietf-ace-oauth-authz].

It can be imagined that an installation is divided in a set of security regions, each one with its own RD(s) to discover the endpoints that are part of a given security region. An endpoint that wants to discover an RD, responsible for a given region, needs to be authorized to learn the contents of a given RD. Within a region, for a given RD, a more fine-grained security division is possible based on the values of the endpoint registration parameters. Authorization to discover endpoints with a given set of filter values is recommended for those cases.

When a node registers its endpoints, criteria are needed to authorize the node to enter them. An important aspect is the uniqueness of the (endpoint name, and optional sector) pair within the RD. Consider the two cases separately: (1) CT registers endpoints, and (2) the registering node registers its own endpoint(s).

- o A CT needs authorization to register a set of endpoints. This authorization can be based on the region, i.e. a given CT is authorized to register any endpoint (endpoint name, sector) into a given RD, or to register an endpoint with (endpoint name, sector) value pairs assigned by the AS, or can be more fine-grained, including a subset of registration parameter values.
- o A given endpoint that registers itself, needs to prove its possession of its unique (endpoint name, sector) value pair. Alternatively, the AS can authorize the endpoint to register with an (endpoint name, sector) value pair assigned by the AS.

A separate document needs to specify these aspects to ensure interoperability between registering nodes and RD. The subsections below give some hints how to handle a subset of the different aspects.

7.1. Secure RD discovery

The Resource Server (RS) discussed in [I-D.ietf-ace-oauth-authz] is equated to the RD. The client (C) needs to discover the RD as discussed in Section 4.1. C can discover the related AS by sending a request to the RD. The RD denies the request by sending the address of the related AS, as discussed in section 5.1 of [I-D.ietf-ace-oauth-authz]. The client MUST send an authorization request to the AS. When appropriate, the AS returns a token that specifies the authorization permission which needs to be specified in a separate document.

7.2. Secure RD filtering

The authorized parameter values for the queries by a given endpoint must be registered by the AS. The AS communicates the parameter values in the token. A separate document needs to specify the parameter value combinations and their storage in the token. The RD decodes the token and checks the validity of the queries of the client.

7.3. Secure endpoint Name assignment

This section only considers the assignment of a name to the endpoint based on an automatic mechanism without use of AS. More elaborate protocols are out of scope. The registering endpoint is authorized by the AS to discover the RD and add registrations. A token is provided by the AS and communicated from registering endpoint to RD. It is assumed that DTLS is used to secure the channel between registering endpoint and RD, where the registering endpoint is the DTLS client. Assuming that the client is provided by a certificate at manufacturing time, the certificate is uniquely identified by the CN field and the serial number. The RD can assign a unique endpoint name by using the certificate identifier as endpoint name. Proof of possession of the endpoint name by the registering endpoint is checked by encrypting the certificate identifier with the private key of the registering endpoint, which the RD can decrypt with the public key stored in the certificate. Even simpler, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD can check for the improbable replication of the random value. The RD MUST check that registering endpoint uses only one random value for each authorized endpoint.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document.

8.1. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Section 7.3 specifies an example that removes this threat for endpoints that have a certificate installed.

8.2. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

8.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. There is also a danger that NTP Servers could become implicated in denial-of-service (DoS) attacks since they run on unprotected UDP, there is no return routability check, and they can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than

requests, RDs can unknowingly become part of a DDoS amplification attack.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats":

- o Resource Directory Address Option (38)

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- o the human readable name of the parameter,
- o the short name as used in query parameters or target attributes,
- o indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,

- o validity requirements if any, and
- o a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	60-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page Count	page count	Integer	L	Used for pagination
Endpoint Type	et	Integer Section 9.3.1	L RLA	Used for pagination Semantic type of the endpoint (see Section 9.4)

Table 2: RD Parameters

(Short: Short name used in query parameters or target attributes.
Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

9.3.1. Full description of the "Endpoint Type" Registration Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, Resource Directory implementations automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- o The values MUST be related to the purpose described in Section 9.3.1.
- o The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- o It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- o "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 - "all CoRE resource directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 - "all CoRE resource directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LWM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
Resource directory	2001:db8:4::ff

Table 3: interface SLAAC addresses

In Section 10.1.2 the use of resource directory during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	light
luminary1	lm_R2-4-015_wndw	/light/middle	light
luminary1	lm_R2-4-015_wndw	/light/right	light
luminary2	lm_R2-4-015_door	/light/left	light
luminary2	lm_R2-4-015_door	/light/middle	light
luminary2	lm_R2-4-015_door	/light/right	light
Presence sensor	ps_R2-4-015_door	/ps	p-sensor

Table 4: Resource Directory identifiers

It is assumed that the CT knows the RD's address, and has performed URI discovery on it that returned a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
    ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"
```

```
Res: 2.01 Created
Location-Path: /rd/4521
```

```
Req: POST coap://[2001:db8:4::ff]/rd
    ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"
```

```
Res: 2.01 Created
Location-Path: /rd/4522
```

```
Req: POST coap://[2001:db8:4::ff]/rd
    ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="p-sensor"
```

```
Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 23: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 24: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
base="coap://[ff05::1]";rt="core.rd-ep"
```

Figure 25: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

```
Req: POST coap://[2001:db8:4::1]/coap-group
Content-Format: application/coap-group+json
Payload:
{ "a": "[ff05::1]", "n": "grp_R2-4-015" }

Res: 2.01 Created
Location-Path: /coap-group/1
```

Figure 26: Example use of direct multicast address configuration

Dependent on the situation, only the address, "a", or the name, "n", is specified in the coap-group resource.

The presence sensor can learn the presence of groups that support resources with rt=light in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP (OMA Name Authority). LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration sectors and does not currently use the rd-lookup interface.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, and applications are currently out of scope for LWM2M.

The location of the LWM2M Server and RD URI path is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD is used. LWM2M Servers and endpoints are not required to implement the /.well-known/core resource.

10.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents

a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

```
{/base-uri}/{/object-id}/{/object-instance}/{/resource-id}/{/resource-instance}
```

The five variables given here are strings. base-uri can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables object-instance, resource-id, and resource-instance can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, object-instance can be "empty" (which is different from "undefined") if resource-id is not "undefined".

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA (OMA Name Authority) registered object ID (0-65535)

object-instance := Object instance identifier (0-65535) or "undefined"/"empty" (see above) to refer to all instances of an object ID

resource-id := OMNA (OMA Name Authority) registered resource ID (0-65535) or "undefined" to refer to all resources within an instance

resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

```
/1/0/1
```

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the

registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

10.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the REST API, described in Section 5. The RD registration URI path of the LWM2M Resource Directory is specified to be "/rd".

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 2 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name, Lifetime, and LWM2M Version are mandatory parameters for the register operation, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Binding Mode	b	{"U", "UQ", "S", "SQ", "US", "UQS"}	Available Protocols
LWM2M Version	ver	1.0	Spec Version
SMS Number	sms		MSISDN

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
base - Registration Base URI

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

```
</1>,</1/0>,</3/0>,</5>
```

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

10.2.3. LWM2M Update Endpoint Registration

The Lwm2M update is really very similar to the registration update as described in Section 5.3.1, with the only difference that there are more parameters defined and available. All the parameters listed in that section are also available with the initial registration but are all optional:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint's UDP port or IP address are changed.

10.2.4. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.3.2.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

changes from -22 to -23

- o Explain that updates can not remove attributes
- o Typo fixes

changes from -21 to -22

- o Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
- o Fix erroneous examples
- o Editorial changes
 - * Add figure numbers to examples
 - * Update RD parameters table to reflect changes of earlier versions in the text
 - * Typos and minor wording

changes from -20 to -21

(Processing comments during WGLC)

- o Defer outdated description of using DNS-SD to find an RD to the defining document
- o Describe operational conditions in automation example
- o Recommend particular discovery mechanisms for some managed network scenarios

changes from -19 to -20

(Processing comments from the WG chair review)

- o Define the permissible characters in endpoint and sector names
- o Express requirements on NAT situations in more abstract terms
- o Shifted heading levels to have the interfaces on the same level
- o Group instructions for error handling into general section
- o Simple Registration: process reflowed into items list
- o Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- o Update acknowledgements
- o Assorted editorial changes
 - * Unify examples style
 - * Terminology: RDAO defined and not only expanded
 - * Add CT to Figure 1
 - * Consistency in the use of the term "Content Format"

changes from -18 to -19

- o link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- o Remove informative references to documents not mentioned any more

changes from -17 to -18

- o Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- o Acknowledging the -17 version as part of the draft
- o Move "Read endpoint links" operation to future specification like PATCH
- o Demote links-json to an informative reference, and removed them from exchange examples
- o Add note on unusability of link-local IP addresses, and describe mitigation.
- o Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- o Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- o Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- o Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD
- o Simplify the rules governing when a registration resource can or must be changed.
- o Drop a figure that has become useless due to the changes of and -13 and -17
- o Wording consistency fixes: Use "Registrations" and "target attributes"
- o Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- o State that the base attribute value is part of endpoint lookup even when implicit in the registration
- o Update references from RFC5988 to its update RFC8288

- o Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- o Removed groups that are enumerations of registrations and have dedicated mechanism
- o Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- o Recommend a common set of resources for members of a group
- o Clarified use of multicast group in lighting example
- o Add note on concurrent registrations from one EP being possible but not expected
- o Refresh web examples appendix to reflect current use of Modernized Link Format
- o Add examples of URIs where Modernized Link Format matters
- o Editorial changes

changes from -14 to -15

- o Rewrite of section "Security policies"
- o Clarify that the "base" parameter text applies both to relative references both in anchor and href
- o Renamed "Registree-EP" to Registrant-EP"
- o Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- o Fixed examples
- o Editorial changes

changes from -13 to -14

- o Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
 - o Introduced resource types core.rd-ep and core.rd-gp
 - o Registration management moved to appendix A, including endpoint and group lookup
 - o Minor editorial changes
 - * PATCH/iPATCH is clearly deferred to another document
 - * Recommend against query / fragment identifier in con=
 - * Interface description lists are described as illustrative
 - * Rewording of Simple Registration
 - o Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
 - o Lookup: href are matched against resolved values (previously, this was unspecified)
 - o Lookup: lt are not exposed any more
 - o con/base: Paths are allowed
 - o Registration resource locations can not have query or fragment parts
 - o Default life time extended to 25 hours
 - o clarified registration update rules
 - o lt-value semantics for lookup clarified.
 - o added template for simple registration
- changes from -12 to -13
- o Added "all resource directory" nodes MC address
 - o Clarified observation behavior
 - o version identification

- o example rt= and et= values
- o domain from figure 2
- o more explanatory text
- o endpoints of a groups hosted by different RD
- o resolve RFC6690-vs-8288 resolution ambiguities:
 - * require registered links not to be relative when using anchor
 - * return absolute URIs in resource lookup

changes from -11 to -12

- o added Content Model section, including ER diagram
- o removed domain lookup interface; domains are now plain attributes of groups and endpoints
- o updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- o improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- o updated LWM2M description
- o clarified where relative references are resolved, and how context and anchor interact
- o new appendix on the interaction with RFCs 6690, 5988 and 3986
- o lookup interface: group and endpoint lookup return group and registration resources as link targets
- o lookup interface: search parameters work the same across all entities
- o removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- o removed plurality definition (was only needed for link modification)
- o enhanced IANA registry text

- o state that lookup resources can be observable
 - o More examples and improved text
- changes from -09 to -10
- o removed "ins" and "exp" link-format extensions.
 - o removed all text concerning DNS-SD.
 - o removed inconsistency in RDAO text.
 - o suggestions taken over from various sources
 - o replaced "Function Set" with "REST API", "base URI", "base path"
 - o moved simple registration to registration section
- changes from -08 to -09
- o clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
 - o changed "ins" ABNF notation.
 - o various editorial improvements, including in examples
 - o clarifications for RDAO
- changes from -07 to -08
- o removed link target value returned from domain and group lookup types
 - o Maximum length of domain parameter 63 bytes for consistency with group
 - o removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
 - o add IPv6 ND Option for discovery of an RD
 - o clarify group configuration section 6.1 that endpoints must be registered before including them in a group
 - o removed all superfluous client-server diagrams

- o simplified lighting example
- o introduced Commissioning Tool
- o RD-Look-up text is extended.

changes from -06 to -07

- o added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- o editorial updates to section 9
- o update author information
- o minor text corrections

Changes from -05 to -06

- o added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- o added Update Endpoint Links using PATCH
- o http access made explicit in interface specification
- o Added http examples

Changes from -03 to -04:

- o Added http response codes
- o Clarified endpoint name usage
- o Add application/link-format+cbor content-format

Changes from -02 to -03:

- o Added an example for lighting and DNS integration
- o Added an example for RD use in OMA LWM2M
- o Added Read Links operation for link inspection by endpoints
- o Expanded DNS-SD section

- o Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- o Added a catalogue use case.
- o Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- o Additional examples section added for more complex use cases.
- o New DNS-SD mapping section.
- o Added text on endpoint identification and authentication.
- o Error code 4.04 added to Registration Update and Delete requests.
- o Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- o Removed the ETag validation feature.
- o Place holder for the DNS-SD mapping section.
- o Explicitly disabled GET or POST on returned Location.
- o New registry for RD parameters.
- o Added support for the JSON Link Format.
- o Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.

- o Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.
- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.
- o Fixed tickets 383 and 372

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.
- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the inclusion of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model---toward a unified view of data", ACM Transactions on Database Systems Vol. 1, pp. 9-36, DOI 10.1145/320434.320440, March 1976.
- [I-D.bormann-t2trg-rel-impl] Bormann, C., "impl-info: A link relation type for disclosing implementation information", draft-bormann-t2trg-rel-impl-00 (work in progress), January 2018.
- [I-D.hartke-t2trg-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-08 (work in progress), March 2019.

- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-10 (work in progress), February 2018.
- [I-D.ietf-core-rd-dns-sd]
Stok, P., Koster, M., and C. Amsuess, "CoRE Resource Directory: DNS-SD mapping", draft-ietf-core-rd-dns-sd-05 (work in progress), July 2019.
- [I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-09 (work in progress), July 2018.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [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/info/rfc8132>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside a Resource Directory.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
</light>;rt="light";if="core.a",
</color-temperature>;if="core.p";u="K"

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.01 Content
Payload:
</rd/501>;ep="GRP_R2-4-015";et="core.rd-group";
      base="coap://[ff05::1]",
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:db8::1]";rt="core.rd-ep"
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```

Req: GET /rd-lookup/res?et=core.rd-group

<coap://[ff35:30:2001:db8::1]/light>;rt="light";if="core.a";
  et="core.rd-group";anchor="coap://[ff35:30:2001:db8::1]",
<coap://[ff35:30:2001:db8::1]/color-temperature>;if="core.p";u="K";
  et="core.rd-group";
  anchor="coap://[ff35:30:2001:db8::1]"

```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines link headers, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in ".well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

The explanation of the steps makes some shortcuts in the more confusing details of [RFC6690], which are justified as all examples being in Limited Link Format.

B.1. A simple example

Let's start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

```

GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0

```

Figure 30: Example of direct resource discovery

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with

Uri-Path: "temp", the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is `"/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/temp` against.

The Base URI of the requested resource can be composed from the header options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/temp"` into `"coap://[2001:db8:f0::1]/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is the URI specified in the GET request `"coap://[2001:db8:f0::1]/.well-known/core"`. A full English expression of the "host relation" is:

`'"coap://[2001:db8:f0::1]/.well-known/core" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'`

B.2. A slightly more complex example

Omitting the `"rt=temperature"` filter, the discovery query would have given some more records in the payload:

```

GET coap://[ff02::fd]:5683/.well-known/core

RES 2.05 Content
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/temp";
  rel="describedby"

```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to "coap://[2001:db8:f0::1]/sensors/temp". That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as "coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t".

Following the same resolution steps, the fourth record can be read as "coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123".

B.3. Enter the Resource Directory

The resource directory tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the resource directory that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/core?ep=simple-host1
```

Figure 32: Example request starting a simple registration

The resource directory would have accepted the registration, and queried the simple host's ".well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 90000 seconds, and the endpoint registration Base URI is "coap://[2001:db8:f0::1]" following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: scheme://authority without path suffix.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and issue a request to "coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature" to receive the following data:

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
  anchor="coap://[2001:db8:f0::1]"
```

Figure 33: Example payload of a response to a resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

```
'"coap://[2001:db8:f0::1]" is hosting the resource
"coap://[2001:db8:f0::1]/temp", which is of the resource type
"temperature" and can be accessed using the text/plain content
format.'
```

(The difference is whether "/" or "/.well-known/core" hosts the resources, which does not matter in this application; if it did, the endpoint would have been more explicit. Actually, /.well-known/core does NOT host the resource but stores a URI reference to the resource.)

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1". A request to "coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1" would return

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
  anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/light>;rt=light-lux;ct=0;
  anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/t>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel="describedby"
```

Figure 34: Extended example payload of a response to a resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/temp>;rt=temperature;ct=0;
  anchor="coap+tcp://simple-host1.example.com"
```

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link headers

While link-format and Link headers look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288], which are dealt with differently:

- o "Resolving the target against the anchor": [RFC6690] Section 2.1 states that the anchor of a link is used as the Base URI against which the term inside the angle brackets (the target) is resolved, falling back to the resource's URI with paths stripped off (its "Origin"). In contrast to that, [RFC8288] Section B.2 describes that the anchor is immaterial to the resolution of the target reference.

RFC6690, in the same section, also states that absent anchors set the context of the link to the target's URI with its path stripped off, while according to [RFC8288] Section 3.2, the context is the resource's base URI.

The rules introduced in Appendix C ensure that an RD does not need to deal with those differences when processing input data. Lookup results are required to be absolute references for the same reason.

- o There is no percent encoding in link-format documents.

A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link headers are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header in a page about a Swedish city might read

```
"Link: </temperature/Malm%C3%B6>;rel="live-environment-data"
```

a link-format document from the same source might describe the link as

```
"</temperature/Malmoe>;rel="live-environment-data"
```

Parsers and producers of link-format and header data need to be aware of this difference.

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of a Resource Directory (e.g. base values with path components).

This appendix describes a subset of link format documents called Limited Link Format. The rules herein are not very limiting in practice – all examples in RFC6690, and all deployments the authors are aware of already stick to them – but ease the implementation of resource directory servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

- o All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash),
- o if the anchor reference starts with a scheme, the target reference starts with a scheme as well (i.e. relative references in target cannot be used when the anchor is a full URI), and
- o the application does not care whether links without an explicitly given anchor have the origin's "/" or "/.well-known/core" resource as their link context.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

Peter van der Stok
consultant

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

Christian Amsuess (editor)
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI
November 4, 2019

SenML Data Value Content-Format Indication
draft-ietf-core-senml-data-ct-01

Abstract

The Sensor Measurement Lists (SenML) media type supports multiple types of values, from numbers to text strings and arbitrary binary data values. In order to simplify processing of the data values this document proposes to specify a new SenML field for indicating the Content-Format of the data.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. SenML Content-Format ("ct") Field	3
4. SenML Base Content-Format ("bct") Field	4
5. Mandatory to Understand Content-Format	4
6. Examples	4
7. Security Considerations	5
8. IANA Considerations	5
Acknowledgements	5
10. References	6
10.1. Normative References	6
10.2. Informative References	6
Authors' Addresses	7

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] can be used to send various different kinds of data. In the example given in Figure 1, a temperature value, an indication whether a lock is open, and a data value (with SenML field "vd") read from an NFC reader is sent in a single SenML pack.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:", "n":"temp", "u":"Cel", "v":7.1},
  {"n":"open", "vb":false},
  {"n":"nfc-reader", "vd":"aGkgCg"}
]
```

Figure 1: SenML pack with unidentified binary data

The receiver is expected to know how to interpret the data in the "vd" field based on the context, e.g., name of the data source and out-of-band knowledge of the application. However, this context may not always be easily available to entities processing the SenML pack. To facilitate automatic interpretation it is useful to be able to indicate an Internet media type and content-coding right in the SenML Record. The CoAP Content-Format (Section 12.3 in [RFC7252]) provides just this information; enclosing a Content-Format number (in this case number 60 as defined for content-type application/cbor in [RFC7049]) in the Record is illustrated in Figure 2. All registered CoAP Content-Formats are listed in the Content-Formats subregistry of the CoRE Parameters registry [IANA.core-parameters].

```
{"n":"nfc-reader", "vd":"gmNmb28YKg", "ct":"60"}
```

Figure 2: SenML Record with binary data identified as CBOR

In this example SenML Record the data value contains a string "foo" and a number 42 encoded in a CBOR [RFC7049] array. Since the example above uses the JSON format of SenML, the data value containing the binary CBOR value is base64-encoded. The data value after base64 decoding is shown with CBOR diagnostic notation in Figure 3.

```
82          # array(2)
 63          # text(3)
    666F6F  # "foo"
 18 2A      # unsigned(42)
```

Figure 3: Example Data Value in CBOR diagnostic notation

2. Terminology

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.

Readers should also be familiar with the terms and concepts discussed in [RFC8428]. Awareness of terminology issues discussed in [I-D.bormann-core-media-content-type-format] can also be very helpful.

3. SenML Content-Format ("ct") Field

When a SenML Record contains a Data Value field ("vd"), the Record MAY also include a Content-Format indication field. The Content-Format indication uses label "ct" and a string value with either a CoAP Content-Format identifier in decimal form with no leading zeros except for the value "0" itself (representing an unsigned integer in the range of 0-65535, similar to the CoRE Link Format [RFC6690] "ct" attribute) or with a string containing a Content-Type and optionally a Content-Coding (see below).

The CoAP Content-Format identifier provides a simple and efficient way to indicate the type of the data. Since some Internet media types and their content coding and parameter alternatives do not have assigned CoAP Content-Format identifiers, using Content-Type and Content-Coding is also allowed. Both methods use a string value in the "ct" field to keep its data type consistent across uses. When

the "ct" field contains only digits, it is interpreted as a CoAP Content-Format identifier.

To indicate that a Content-Coding is used with a Content-Type, the Content-Coding value (e.g., "deflate" [RFC7230]) is appended to the Content Type (media type and parameters, if any), separated by a "@" sign. For example: "text/plain; charset=utf-8@deflate". If Content-Coding is not specified with a Content-Type (no "@" sign is present outside any media type parameters), the identity (i.e., no) transformation is used.

4. SenML Base Content-Format ("bct") Field

The Base Content-Format Field, label "bct", provides a default value for the Content-Format Field (label "ct") within its range. The range of the base field includes the Record containing it, up to (but not including) the next Record containing a "bct" field, if any, or up to the end of the pack otherwise. Resolution (Section 4.6 of [RFC8428]) of this base field is performed by adding its value with the label "ct" to all Records in this range that carry a "vd" field but do not already contain a Content-Format ("ct") field.

5. Mandatory to Understand Content-Format

If the Content-Format field needs to be understood by all processors of the SenML Pack, the mandatory to understand versions of the fields, "ct_" and "bct_", can be used. These fields have identical semantics to the "ct" and "bct" fields respectively except that a SenML processor that does not support this specification would reject a SenML Pack with such fields and generate an error (see Section 4.4 of [RFC8428]).

Using the regular Content-Format indication enables to use this extension in a backward compatible way to indicate information that is not critical to be understood. The choice between the two methods is application dependent.

If both a "ct_" field and a "ct" field are present in a resolved Record (i.e., from fields in the Record or from base fields), the "ct_" field overrides the "ct" field. Using both "ct" and "ct_" in the same Record is NOT RECOMMENDED as it MAY be treated as an error by the recipient.

6. Examples

The following examples are valid values for the "ct" and "bct" fields (explanation/comments in parenthesis):

- o "60" (CoAP Content-Format for "application/cbor")
- o "0" (CoAP Content-Format for "text/plain" with parameter "charset=utf-8")
- o "application/json" (JSON Content-Type - equivalent to "50" CoAP Content-Format identifier)
- o "application/json@deflate" (JSON Content-Type with "deflate" as Content-Coding - equivalent to "11050" CoAP Content-Format identifier)
- o "text/csv" (Comma-Separated Values (CSV) [RFC4180] Content-Type)
- o "text/csv@gzip" (CSV with "gzip" as Content-Coding)

7. Security Considerations

The indication of a media type in the data does not exempt a consuming application from properly checking its inputs. Also, the ability for an attacker to supply crafted SenML data that specify media types chosen by the attacker may expose vulnerabilities of handlers for these media types to the attacker.

8. IANA Considerations

(Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification and remove this note.)

IANA is requested to assign new labels in the "SenML Labels" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]) for the Content-Format indication as per Table 1:

Name	Label	JSON Type	XML Type	Reference
Base Content-Format	bct	String	string	RFC-AAAA
Content-Format	ct	String	string	RFC-AAAA

Table 1: IANA Registration for new SenML Labels

Acknowledgements

The authors would like to thank Sergio Abreu for the discussions leading to the design of this extension and Isaac Rivera for reviews and feedback.

10. References

10.1. Normative References

- [IANA.senml]
IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

10.2. Informative References

- [I-D.bormann-core-media-content-type-format]
Bormann, C., "On Media-Types, Content-Types, and related terminology", draft-bormann-core-media-content-type-format-01 (work in progress), July 2019.
- [IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters",
<<http://www.iana.org/assignments/core-parameters>>.
- [RFC4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <<https://www.rfc-editor.org/info/rfc4180>>.

- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 18, 2020

A. Keranen
Ericsson
M. Mohajer
u-blox UK
August 17, 2019

FETCH & PATCH with Sensor Measurement Lists (SenML)
draft-ietf-core-senml-etch-05

Abstract

The Sensor Measurement Lists (SenML) media type and data model can be used to send collections of resources, such as batches of sensor data or configuration parameters. The CoAP iPATCH, PATCH, and FETCH methods enable accessing and updating parts of a resource or multiple resources with one request. This document defines new media types for the CoAP iPATCH, PATCH, and FETCH methods for resources represented with the SenML data model.

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 February 18, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Using FETCH and (i)PATCH with SenML	3
3.1. SenML FETCH	4
3.2. SenML (i)PATCH	4
4. Fragment Identification	5
5. Security Considerations	6
6. IANA Considerations	6
6.1. CoAP Content-Format Registration	6
6.2. senml-etch+json Media Type	6
6.3. senml-etch+cbor Media Type	7
7. Acknowledgements	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
Authors' Addresses	10

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] and data model can be used to transmit collections of resources, such as batches of sensor data or configuration parameters.

An example of a SenML collection is shown below:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":true},
  {"n":"5851", "v":42},
  {"n":"5750", "vs":"Ceiling light"}
]
```

Here three resources "3311/0/5850", "3311/0/5851", and "3311/0/5750", of an IPSO dimmable light smart object [IPSO] are represented using a single SenML Pack with three SenML Records. All resources share the same base name "2001:db8::2/3311/0/", hence full names for resources are "2001:db8::2/3311/0/5850", etc.

The CoAP [RFC7252] iPATCH, PATCH, and FETCH methods [RFC8132] enable accessing and updating parts of a resource or multiple resources with one request.

This document defines two new media types, one using the JavaScript Object Notation (JSON) [RFC8259] and one using the Concise Binary Object Representation (CBOR) [RFC7049], that can be used with the CoAP iPATCH, PATCH, and FETCH methods for resources represented with the SenML data model. The semantics of the new media types are the same for the CoAP PATCH and iPATCH methods. The rest of the document uses term "(i)PATCH" when referring to both methods.

2. Terminology

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.

Readers should also be familiar with the terms and concepts discussed in [RFC8132] and [RFC8428]. Also the following terms are used in this document:

Fetch Record: One set of parameters that is used to match SenML Record(s).

Fetch Pack: One or more Fetch Records in an array structure.

Patch Record: One set of parameters similar to Fetch Record but also containing instructions on how to change existing SenML Pack(s).

Patch Pack: One or more Patch Records in an array structure.

Target Record: A Record in a SenML Pack that is matching the selection criteria of a Fetch or Patch Record and hence is a target for a Fetch or Patch operation.

(i)PATCH: A term that refers to both CoAP "PATCH" and "iPATCH" methods when there is no difference in this specification in which one is used.

3. Using FETCH and (i)PATCH with SenML

The FETCH/(i)PATCH media types for SenML are modeled as extensions to the SenML media type to enable re-use of existing SenML parsers and generators, in particular on constrained devices. Unless mentioned otherwise, FETCH and PATCH Packs are constructed with the same rules and constraints as SenML Packs.

The key difference to the SenML media type is allowing the use of a "null" value for removing records with the (i)PATCH method. Also the

Fetch and Patch Records do not have default time or base version when the fields are omitted.

3.1. SenML FETCH

The FETCH method can be used to select and return a subset of records, in sequence, of one or more SenML Packs. The SenML Records are selected by giving a set of names that, when resolved, match resolved names in a SenML Pack. The names for a Fetch Pack are given using the SenML "name" and/or "base name" Fields. The names are resolved by concatenating the base name with the name field as defined in [RFC8428].

For example, to select the IPSO resources "5850" and "5851" from the example in Section 1, the following Fetch Pack can be used:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850"},
  {"n":"5851"}
]
```

The result to a FETCH request with the example above would be:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":true},
  {"n":"5851", "v":42},
]
```

When SenML Records contain also time values, a name may no longer uniquely identify a single Record. When no time is given in a Fetch Record, all SenML Records with the given name are matched (i.e., unlike with SenML Records, lack of time field in a Fetch Record does not imply time value zero). When time is given in the Fetch Record, only a SenML Record (if any) with equal resolved time value and name is matched.

The resolved form of records (Section 4.6 of [RFC8428]) is used when comparing the names and times of the Target and Fetch Records to accommodate for differences in use of the base values.

3.2. SenML (i)PATCH

The (i)PATCH method can be used to change the values of SenML Records, to add new Records, and to remove existing Records. The names and times of the Patch Records are given and matched in same way as for the Fetch Records, except each Patch Record can match at most one Target Record. Patch Packs can also include new values and

other SenML Fields for the Records. Application of Patch Packs is idempotent.

When the name in a Patch Record matches with the name in an existing Record, the resolved time values are compared. If the time values either do not exist in both Records or are equal, the Target Record is replaced with the contents of the Patch Record.

If a Patch Record contains a name, or combination of a time value and a name, that do not exist in any existing Record in the Pack, the given Record, with all the fields it contains, is added to the Pack.

If a Patch Record has a value ("v") field with value null, the matched Record (if any) is removed from the Pack.

For example, the following document could be given as an (i)PATCH payload to change/set values of two SenML Records for the example in Section 1:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":false},
  {"n":"5851", "v":10}
]
```

If the request is successful, the resulting representation of the example SenML Pack would be as follows:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":false},
  {"n":"5851", "v":10},
  {"n":"5750", "vs":"Ceiling light"}
]
```

As another example, the following document could be given as an (i)PATCH payload to remove the two SenML Records:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "v":null},
  {"n":"5851", "v":null}
]
```

4. Fragment Identification

Fragment identification is supported by analogously applying fragment identifiers as specified in Section 9 of [RFC8428] to the Fetch/Patch Records.

5. Security Considerations

The security and privacy considerations of SenML apply also with the FETCH and (i)PATCH methods.

In FETCH and (i)PATCH requests, the client can pass arbitrary names to the target resource for manipulation. The resource implementer must take care to only allow access to names that are actually part of (or accessible through) the target resource.

If the client is not allowed to do a GET or PUT on the full target resource (and thus all the names accessible through it), access control rules must be evaluated for each record in the pack.

6. IANA Considerations

This document registers two new media types and CoAP Content-Format IDs for both media types.

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this document.

6.1. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML PATCH and FETCH media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. The assigned IDs are shown in Table 1.

Media type	Encoding	ID
application/senml-etch+json	-	TBD-320
application/senml-etch+cbor	-	TBD-322

Table 1: CoAP Content-Format IDs

6.2. senml-etch+json Media Type

Type name: application

Subtype name: senml-etch+json

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: See Section 5 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: Applications that use the SenML media type for resource representation.

Fragment identifier considerations: Fragment identification for application/senml-etch+json is supported by using fragment identifiers as specified by RFC AAAA.

Additional information:

Magic number(s): none

File extension(s): senml-etchj

Windows Clipboard Name: "SenML FETCH/PATCH format"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-etch-json
conforms to public.text

Person & email address to contact for further information: Ari
Keranen ari.keranen@ericsson.com

Intended usage: COMMON

Restrictions on usage: None

Author: Ari Keranen ari.keranen@ericsson.com

Change controller: IESG

6.3. senml-etch+cbor Media Type

Type name: application

Subtype name: senml-etch+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations: See Section 5 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: Applications that use the SenML media type for resource representation.

Fragment identifier considerations: Fragment identification for application/senml-etch+cbor is supported by using fragment identifiers as specified by RFC AAAA.

Additional information:

Magic number(s): none

File extension(s): senml-etchc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-etch-cbor
conforms to public.data

Person & email address to contact for further information: Ari
Keranen ari.keranen@ericsson.com

Intended usage: COMMON

Restrictions on usage: None

Author: Ari Keranen ari.keranen@ericsson.com

Change controller: IESG

7. Acknowledgements

The use of FETCH and (i)PATCH methods with SenML was first introduced by the OMA SpecWorks LwM2M v1.1 specification. This document generalizes the use to any SenML representation. The authors would like to thank Carsten Bormann, Christian Amsuess, Jaime Jimenez, Klaus Hartke, Michael Richardson, and other participants from the IETF CoRE and OMA SpecWorks DMSE working groups who have contributed ideas and reviews.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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/info/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/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

8.2. Informative References

[IPSO] IPSO, "IPSO Light Control Smart Object", 2018,
<[http://www.openmobilealliance.org/tech/profiles/
lwm2m/3311.xml](http://www.openmobilealliance.org/tech/profiles/lwm2m/3311.xml)>.

Authors' Addresses

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Mojan Mohajer
u-blox UK

Email: Mojan.Mohajer@u-blox.com

Network Working Group
Internet-Draft
Updates: 8428 (if approved)
Intended status: Standards Track
Expires: May 7, 2020

C. Bormann
Universitaet Bremen TZI
November 04, 2019

Additional Units for SenML
draft-ietf-core-senml-more-units-03

Abstract

The Sensor Measurement Lists (SenML) media type supports the indication of units for a quantity represented. This short document registers a number of additional unit names in the IANA registry for Units in SenML. It also defines a registry for secondary units that cannot be in SenML's main registry as they are derived by linear transformation from units already in that registry; RFC 8428 is updated to also accept these units.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. New Units	3
3. Rationale	3
4. New Registry	4
5. Security Considerations	6
6. IANA Considerations	6
Acknowledgements	6
8. References	6
8.1. Normative References	6
8.2. Informative References	7
Author's Address	7

1. Introduction

The Sensor Measurement Lists (SenML, [RFC8428]) media type supports the indication of a unit, using the SenML field "u", for the quantity given as a data value in a SenML record. For this purpose, SenML defines an IANA registry of defined Unit names and their meanings; in the present document, we call the Unit names registered there "primary Unit names".

This short document registers a number of additional units in the IANA registry for Units in SenML that appear to be necessary for further adopting SenML in other Standards Development Organizations (SDOs).

The document also defines a registry for secondary Unit names that cannot be in SenML's main registry as they are derived by linear transformation from units already in that registry. [RFC8428] is updated to also accept these secondary Unit names in place of the (primary) Unit names defined in [RFC8428].

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. New Units

IANA is requested to assign new units in the "SenML Units" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]):

Symbol	Description	Type	Reference
B	Byte (information content)	float	RFCthis
VA	volt-ampere (Apparent Power)	float	RFCthis
VAs	volt-ampere second (Apparent Energy)	float	RFCthis
var	volt-ampere reactive (Reactive Power)	float	RFCthis
vars	volt-ampere reactive second (Reactive Energy)	float	RFCthis
J/m	joule per meter (Energy per distance)	float	RFCthis
kg/m3	kilogram per cubic meter (mass concentration)	float	RFCthis
deg	degree (angle)*	float	RFCthis

Table 1: New units registered for SenML

3. Rationale

SenML [RFC8428] takes the position that unscaled SI units should always be used. However, SenML makes one exception: The degree Celsius (as Cel) is allowed as an alternative to the K (Kelvin).

This document takes the position that the same should apply to a small number of alternative units in wide use:

- o The Byte. [IEC-80000-13] defines both the bit (item 13-9.b) and the byte (item 13-9.c, also called octet) as alternative names for the coherent unit one for the purpose of giving storage capacity and related quantities. While the name octet is associated with the symbol o, this is in wide use only in French-speaking countries. Globally more wide-spread is the symbol B for byte, even though B is already taken in SI for bel. [RFC8428] therefore registers dB as the SenML unit for logarithmic relative power, leaving B free for the usage proposed here. While this is potentially confusing, the situation is widely understood in engineering circles and is unlikely to cause actual problems.
- o The Volt-Ampere. [IEC-80000-6] item 6-57.a defines the VA (volt ampere) as a unit for apparent power; items 6-59.a, 6-60.a and

6-61.a also use the unit for complex, reactive, and non-active power.

- o The Volt-Ampere-reactive. [IEC-80000-6] item 6-60.b defines the var (volt ampere reactive) as an alternative (and fully equivalent) unit to VA specifically for reactive power (with the primary unit VA). It is not presently known to this author how the upcoming revision of IEC 80000-6 will update this, but it has become clear that there is strong interest in using this unit specifically for the imaginary content of complex power, reactive power [IEEE-1459].

The unit "degrees" is in wide use in practice for plane angles (as in heading, bearing, etc.). It is marked with an asterisk because the preferred coherent SI unit is radian ("rad").

The Joule per meter is not a traditional electromagnetic unit. It and its scaled derivatives (in particular Wh/km) are used to describe the energy expended for achieving motion over a given distance, e.g., as an equivalent for electrical cars of the inverse of "mileage".

4. New Registry

IANA is requested to create a "secondary units" subregistry in the SenML registry [IANA.senml] defined in [RFC8428].

The registry has six columns:

- o secondary unit: a newly registered name allocated within the same namespace as SenML units
- o description: short description (usually just expansion of abbreviation)
- o SenML unit: an existing SenML unit from the SenML units registry
- o scale, offset: two rational numbers, expressed in decimal (optionally, with a decimal exponent given) or as a fraction divided by a "/" character.
- o Reference: where does the entry come from.

Quantities expressed in the secondary unit can be converted into the SenML unit by first multiplying their value with the scale number and then adding the offset, yielding the value in the given SenML unit.

The initial content of the secondary units registry is provided in Table 2:

secondary unit	description	SenML unit	scale	off set	reference
ms	millisecond	s	1/1000	0	RFCthis
min	minute	s	60	0	RFCthis
h	hour	s	3600	0	RFCthis
kW	kilowatt	W	1000	0	RFCthis
kVA	kilovolt-ampere	VA	1000	0	RFCthis
kvar	kilovar	var	1000	0	RFCthis
Ah	ampere-hour	C	3600	0	RFCthis
Wh	watt-hour	J	3600	0	RFCthis
kWh	kilowatt-hour	J	3600000	0	RFCthis
varh	var-hour	vars	3600	0	RFCthis
kvarh	kilovar-hour	vars	3600000	0	RFCthis
kVAh	kilovolt-ampere-hour	VAs	3600000	0	RFCthis
Wh/km	watt-hour per kilometer	J/m	3.6	0	RFCthis
KiB	kibibyte	B	1024	0	RFCthis
mV	millivolt	V	1/1000	0	RFCthis
mA	milliampere	A	1/1000	0	RFCthis
dBm	decibel (milliwatt)	dBW	1	-30	RFCthis
ug/m3	microgram per cubic meter	kg/m3	1e-9	0	RFCthis
mm/h	millimeter per hour	m/s	1/3600000	0	RFCthis
ppm	parts per million	/	1e-6	0	RFCthis
/100	percent (Note 1)	/	1/100	0	RFCthis
/1000	permille	/	1/1000	0	RFCthis
hPa	hectopascal	Pa	100	0	RFCthis
mm	millimeter	m	1/1000	0	RFCthis
km	kilometer	km	1000	0	RFCthis
km/h	kilometer per hour	m/s	1/3.6	0	RFCthis

Table 2: Secondary units registered for SenML

Note 1: This registration does not use the obvious name "%" because this name has been taken in [RFC8428] already, where it is a NOT RECOMMENDED synonym for "/" (unity) for legacy reasons. Note that the presence of both "%" and "/100" with different meanings is likely to create confusion, so the present document adds some weight to the recommendation against using the counterintuitive unit name "%".

Example: the value of a quantity given as 100 ms is first multiplied by 1/1000, yielding the number 0.1, and then the offset 0 is added, yielding the number 0.1 again, leading to a quantity of 0.1 s. The value of a quantity given as 10 dBm is first multiplied by 1, yielding the number 10, and then the offset -30 is added, yielding the number -20, leading to a quantity of -20 dBW.

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment, with the same guidelines as used for SenML units (Section 12.1 of [RFC8428]), but without applying the rules 4 and 5. Guidelines to the difference between units (which can go into the registry) and quantities are widely available, see for instance [RS] and [BIPM].

SenML packs MAY, but SHOULD NOT, use secondary units in place of SenML units, where the exception of the "SHOULD NOT" lies in the context of specific existing data models that are based on these secondary units.

5. Security Considerations

The security considerations of [RFC8428] apply. The introduction of new measurement units poses no additional security considerations except from a possible potential for additional confusion about the proper unit to use.

6. IANA Considerations

See Section 2 and Section 4.

Acknowledgements

Ari Keranen pointed out the need for additional units in SenML. Comments provided by him as well as by Thomas Fossati, Joaquin Prado, and Jaime Jimenez helped improve the document.

8. References

8.1. Normative References

[IANA.senml]

IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.

[IEC-80000-13]

"Quantities and units - Part 13: Information science and technology", IEC 80000-13, Edition 1.0, March 2008.

- [IEC-80000-6] "Quantities and units - Part 6: Electromagnetism", IEC 80000-6, Edition 1.0, March 2008.
- [IEEE-1459] "IEEE Standard Definitions for the Measurement of Electric Power Quantities Under Sinusoidal, Nonsinusoidal, Balanced, or Unbalanced Conditions", IEEE Std 1459-2010, March 2010.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

8.2. Informative References

- [BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI), 9th edition", 2019, <<https://www.bipm.org/utis/common/pdf/si-brochure/SI-Brochure-9-EN.pdf>>.
- [RS] Rohde&Schwarz, "Standard-compliant usage of quantities, units and equations", version 4.0, November 2016, <https://karriere.rohde-schwarz.de/fileadmin/user_upload/Standard-compliant_usage_of_quantities_units_and_equations_bro_en_5214-5061-62_v0400_96dpi.pdf>.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
I. Petrov, Ed.
Acklio
July 08, 2019

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-07

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned integers used to identify YANG items. This document defines the semantics, the registration, and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

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 January 9, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	3
3. ".sid" file lifecycle	4
4. ".sid" file format	5
5. Security Considerations	9
6. IANA Considerations	9
6.1. Register SID File Format Module	9
6.2. Create new IANA Registry: "SID Mega-Range" registry . . .	9
6.2.1. Structure	9
6.2.2. Allocation policy	10
6.2.2.1. First allocation	10
6.2.2.2. Consecutive allocations	11
6.2.3. Initial contents of the Registry	11
6.3. Create a new IANA Registry: IETF SID Range Registry (managed by IANA)	11
6.3.1. Structure	11
6.3.2. Allocation policy	11
6.3.3. Initial contents of the registry	12
6.4. Create new IANA Registry: "IETF SID Registry"	13
6.4.1. Structure	13
6.4.2. Allocation policy	14
6.4.3. Initial contents of the registry	14
7. Acknowledgments	14
8. References	14
8.1. Normative References	14
8.2. Informative References	15
Appendix A. ".sid" file example	16
Appendix B. SID auto generation	25
Appendix C. ".sid" file lifecycle	25
C.1. SID File Creation	26
C.2. SID File Update	27
Authors' Addresses	28

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called SID, is encoded using a 64-bit unsigned integer. The following items are identified using SIDs:

- o identities
- o data nodes (Note: including those parts of a YANG template as defined by the 'yang-data' extension.)
- o RPCs and associated input(s) and output(s)
- o actions and associated input(s) and output(s)
- o notifications and associated information
- o YANG modules, submodules and features

SIDs are globally unique integers, a registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges".

Assignment of SIDs to YANG items can be automated. For more details how this could be achieved, please consult Appendix B.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned.

Section 3 provides more details about the registration process of YANG modules and associated SIDs. To enable the implementation of this registry, Section 4 defines a standard file format used to store and publish SIDs.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950]:

- o action
- o feature
- o module
- o notification
- o RPC

- o schema node
- o schema tree
- o submodule

The following term is defined in [RFC8040]:

- o yang-data extension

This specification also makes use of the following terminology:

- o item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- o path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/ietf-system:system-state/clock/current-datetime")
- o YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. ".sid" file lifecycle

YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoRECONF [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

Many YANG modules are not created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. In order to do that, they should first obtain a SID range from a registry and use that range to assign or generate SIDs to items of their YANG module. For example how this could be achieved, please refer to Appendix C.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module. Different registries might have different requirements for the registration and publication of the ".sid" files. For diagram of one

of the possibilities, please refer to the activity diagram on Figure 1 in Appendix C.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the 'assignment-ranges' as defined in Section 4. These extra SIDs are used for subsequent assignments.

For an example of this update process, see activity diagram Figure 2 in Appendix C.

4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [RFC7951].

```
<CODE BEGINS> file "ietf-sid-file@2017-11-26.yang"
module ietf-sid-file {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF Core Working Group";

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

    Andy Bierman
    <mailto:andy@yumaworks.com>

    Alexander Pelov
    <mailto:a@ackl.io>";

  description
    "This module defines the structure of the .sid files.

    Each .sid file contains the mapping between the different
    string identifiers defined by a YANG module and a
```

```
    corresponding numeric value called SID.";

revision 2017-11-26 {
    description
        "Initial revision.";
    reference
        "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

typedef revision-identifier {
    type string {
        pattern '\d{4}-\d{2}-\d{2}';
    }
    description
        "Represents a date in YYYY-MM-DD format.";
}

typedef sid {
    type uint64;
    description
        "YANG Schema Item iDentifier";
    reference
        "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

typedef schema-node-path {
    type string {
        pattern
            '(/[a-zA-Z_][a-zA-Z0-9\-\_\.]*:[a-zA-Z_][a-zA-Z0-9\-\_\.]*)' +
            '(/[a-zA-Z_][a-zA-Z0-9\-\_\.]*(:[a-zA-Z_][a-zA-Z0-9\-\_\.]*)?)?)*';
    }
    description
        "Identifies a schema-node path string for use in the
        SID registry. This string format follows the rules
        for an instance-identifier, as defined in RFC 7959,
        except that no predicates are allowed.

        This format is intended to support the YANG 1.1 ABNF
        for a schema node identifier, except module names
        are used instead of prefixes, as specified in RFC 7951.";
    reference
        "RFC 7950, The YANG 1.1 Data Modeling Language;
        Section 6.5: Schema Node Identifier;
        RFC 7951, JSON Encoding of YANG Data;
        Section 6.11: The instance-identifier type";
}

leaf module-name {
```

```
    type yang:yang-identifier;
    description
      "Name of the YANG module associated with this .sid file.";
  }

  leaf module-revision {
    type revision-identifier;
    description
      "Revision of the YANG module associated with this .sid file.
      This leaf is not present if no revision statement is
      defined in the YANG module.";
  }

  list assignment-ranges {
    key "entry-point";
    description
      "SID range(s) allocated to the YANG module identified by
      'module-name' and 'module-revision'.";

    leaf entry-point {
      type sid;
      mandatory true;
      description
        "Lowest SID available for assignment.";
    }

    leaf size {
      type uint64;
      mandatory true;
      description
        "Number of SIDs available for assignment.";
    }
  }

  list items {
    key "namespace identifier";
    description
      "Each entry within this list defined the mapping between
      a YANG item string identifier and a SID. This list MUST
      include a mapping entry for each YANG item defined by
      the YANG module identified by 'module-name' and
      'module-revision'.";

    leaf namespace {
      type enumeration {
        enum module {
          value 0;
          description

```

```
        "All module and submodule names share the same
        global module identifier namespace.";
    }
    enum identity {
        value 1;
        description
            "All identity names defined in a module and its
            submodules share the same identity identifier
            namespace.";
    }
    enum feature {
        value 2;
        description
            "All feature names defined in a module and its
            submodules share the same feature identifier
            namespace.";
    }
    enum data {
        value 3;
        description
            "The namespace for all data nodes, as defined in YANG.";
    }
}
description
    "Namespace of the YANG item for this mapping entry.";
}

leaf identifier {
    type union {
        type yang:yang-identifier;
        type schema-node-path;
    }
    description
        "String identifier of the YANG item for this mapping entry.

        If the corresponding 'namespace' field is 'module',
        'feature', or 'identity', then this field MUST
        contain a valid YANG identifier string.

        If the corresponding 'namespace' field is 'data',
        then this field MUST contain a valid schema node
        path.";
}

leaf sid {
    type sid;
    mandatory true;
    description
```

```
        "SID assigned to the YANG item for this mapping entry.";
    }
}
}
<CODE ENDS>
```

5. Security Considerations

This document defines a new type of identifier used to encode data models defined in YANG [RFC7950]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

6. IANA Considerations

6.1. Register SID File Format Module

This document registers one YANG module in the "YANG Module Names" registry [RFC6020]:

- o name: ietf-sid-file
- o namespace: urn:ietf:params:xml:ns:yang:ietf-sid-file
- o prefix: sid
- o reference: [[THISRFC]]

6.2. Create new IANA Registry: "SID Mega-Range" registry

The name of this registry is "SID Mega-Range". This registry is used to record the delegation of the management of a block of SIDs to third parties (such as SDOs or registrars).

6.2.1. Structure

Each entry in this registry must include:

- o The entry point (first SID) of the registered SID block.
- o The size of the registered SID block. The size MUST be one million (1 000 000) SIDs.
- o The contact information of the requesting organization including:
 - * The policy of SID range allocations: Public, Private or Both.
 - * Organization name

- * URL

The information associated to the Organization name should not be publicly visible in the registry, but should be available. This information includes contact email and phone number and change controller email and phone number.

6.2.2. Allocation policy

The IANA policy for future additions to this registry is "Expert Review" [RFC8126].

An organization requesting to manage a SID Range (and thus have an entry in the SID Mega-Range Registry), must ensure the following capacities:

- o The capacity to manage and operate a SID Range Registry. A SID Range Registry MUST provide the following information for all SID Ranges allocated by the Registry:
 - * Entry Point of allocated SID Range
 - * Size of allocated SID Range
 - * Type: Public or Private
 - + Public Ranges MUST include at least a reference to the YANG module and ".sid" files for that SID Range.
 - + Private Ranges MUST be marked as "Private"
- o A Policy of allocation, which clearly identifies if the SID Range allocations would be Private, Public or Both.
- o Technical capacity to ensure the sustained operation of the registry for a period of at least 5 years. If Private Registrations are allowed, the period must be of at least 10 years.

6.2.2.1. First allocation

For a first allocation to be provided, the requesting organization must demonstrate a functional registry infrastructure.

6.2.2.2. Consecutive allocations

On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

If that extra-allocation is done within 3 years from the last allocation, the experts need to discuss this request on the CORE working group mailing list and consensus needs to be obtained before allocating a new Mega-Range.

6.2.3. Initial contents of the Registry

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Allocation	Organization name	URL
0	1000000	Public	IANA	iana.org

6.3. Create a new IANA Registry: IETF SID Range Registry (managed by IANA)

6.3.1. Structure

Each entry in this registry must include:

- o The SID range entry point.
- o The SID range size.
- o The YANG module name.
- o Document reference.

6.3.2. Allocation policy

The first million SIDs assigned to IANA is sub-divided as follows:

- o The range of 0 to 999 (size 1000) is "Reserved" as defined in [RFC8126].
- o The range of 1000 to 59,999 (size 59,000) is reserved for YANG modules defined in RFCs. The IANA policy for additions to this registry is "Expert Review" [RFC8126].

- * The Expert MUST verify that the YANG module for which this allocation is made has an RFC (existing RFC) OR is on track to become RFC (early allocation with a request from the WG chairs).
- o The SID range allocated for a YANG module can follow in one of the four categories:
 - * SMALL (50 SIDs)
 - * MEDIUM (100 SIDs)
 - * LARGE (250 SIDs)
 - * CUSTOM (requested by the YANG module author, with a maximum of 1000 SIDs). In all cases, the size of a SID range assigned to a YANG module should be at least 33% above the current number of YANG items. This headroom allows assignment within the same range of new YANG items introduced by subsequent revisions. A larger SID range size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an additional SID range can be allocated to an existing YANG module if the initial range is exhausted.
- o The range of 60,000 to 99,999 (size 40,000) is reserved for experimental YANG modules. This range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability. The IANA policy for this range is "Experimental use" [RFC8126].
- o The range of 100,000 to 999,999 (size 900,000) is "Reserved" as defined in [RFC8126].

Entry Point	Size	IANA policy
0	1,000	Reserved
1,000	59,000	Expert Review
60,000	40,000	Experimental use
100,000	900,000	Reserved

6.3.3. Initial contents of the registry

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Document reference
1000	100	ietf-comi	[I-D.ietf-core-comi]
1100	50	ietf-yang-types	[RFC6991]
1150	50	ietf-inet-types	[RFC6991]
1200	50	iana-crypt-hash	[RFC7317]
1250	50	ietf-netconf-acm	[RFC8341]
1300	50	ietf-sid-file	RFCXXXX
1500	100	ietf-interfaces	[RFC8343]
1600	100	ietf-ip	[RFC8344]
1700	100	ietf-system	[RFC7317]
1800	400	iana-if-type	[RFC7224]

// RFC Ed.: replace XXXX with RFC number assigned to this draft.

For allocation, RFC publication of the YANG module is required as per [RFC8126]. The YANG module must be registered in the "YANG module Name" registry according to the rules specified in section 14 of [RFC6020].

6.4. Create new IANA Registry: "IETF SID Registry"

The name of this registry is "IETF SID Registry". This registry is used to record the allocation of SIDs for individual YANG module items.

6.4.1. Structure

Each entry in this registry must include:

- o The YANG module name. This module name must be present in the "Name" column of the "YANG Module Names" registry.
- o A link to the associated ".yang" file. This file link must be present in the "File" column of the "YANG Module Names" registry.
- o The link to the ".sid" file which defines the allocation.
- o The number of actually allocated SIDs in the ".sid" file.

The ".sid" file is stored by IANA.

6.4.2. Allocation policy

The allocation policy is Expert review. The Expert MUST ensure that the following conditions are met:

- o The ".sid" file has a valid structure:
 - * The ".sid" file MUST be a valid JSON file following the structure of the module defined in RFCXXXX (RFC Ed: replace XXX with RFC number assigned to this draft).
- o The ".sid" file allocates individual SIDs ONLY in the SID Ranges for this YANG module (as allocated in the IETF SID Range Registry):
 - * All SIDs in this ".sid" file MUST be within the ranges allocated to this YANG module in the "IETF SID Range Registry".
- o If another ".sid" file has already allocated SIDs for this YANG module (e.g. for older or newer versions of the YANG module), the YANG items are assigned the same SIDs as in the the other ".sid" file.
- o SIDs never change.

6.4.3. Initial contents of the registry

None.

7. Acknowledgments

The authors would like to thank Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain, Randy Turner and Peter van der Stok for their help during the development of this document and their useful comments during the review process.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-05 (work in progress), May 2019.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- o ietf-system@2014-08-06.yang
- o ietf-yang-types@2013-07-15.yang
- o ietf-inet-types@2013-07-15.yang
- o ietf-netconf-acm@2012-02-22.yang
- o iana-crypt-hash@2014-04-04.yang

```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-system",
      "sid": 1700
    }
  ],
}
```

```
{
  "namespace": "identity",
  "identifier": "authentication-method",
  "sid": 1701
},
{
  "namespace": "identity",
  "identifier": "local-users",
  "sid": 1702
},
{
  "namespace": "identity",
  "identifier": "radius",
  "sid": 1703
},
{
  "namespace": "identity",
  "identifier": "radius-authentication-type",
  "sid": 1704
},
{
  "namespace": "identity",
  "identifier": "radius-chap",
  "sid": 1705
},
{
  "namespace": "identity",
  "identifier": "radius-pap",
  "sid": 1706
},
{
  "namespace": "feature",
  "identifier": "authentication",
  "sid": 1707
},
{
  "namespace": "feature",
  "identifier": "dns-udp-tcp-port",
  "sid": 1708
},
{
  "namespace": "feature",
  "identifier": "local-users",
  "sid": 1709
},
{
  "namespace": "feature",
  "identifier": "ntp",
```

```
    "sid": 1710
  },
  {
    "namespace": "feature",
    "identifier": "ntp-udp-port",
    "sid": 1711
  },
  {
    "namespace": "feature",
    "identifier": "radius",
    "sid": 1712
  },
  {
    "namespace": "feature",
    "identifier": "radius-authentication",
    "sid": 1713
  },
  {
    "namespace": "feature",
    "identifier": "timezone-name",
    "sid": 1714
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:set-current-datetime",
    "sid": 1715
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:set-current-datetime/
      current-datetime",
    "sid": 1716
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system",
    "sid": 1717
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system-restart",
    "sid": 1718
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system-shutdown",
    "sid": 1719
  },
}
```

```
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state",
  "sid": 1720
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock",
  "sid": 1721
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock/boot-datetime",
  "sid": 1722
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock/
    current-datetime",
  "sid": 1723
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform",
  "sid": 1724
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/machine",
  "sid": 1725
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-name",
  "sid": 1726
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-release",
  "sid": 1727
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-version",
  "sid": 1728
},
{
  "namespace": "data",
```



```
    "identifier": "/ietf-system:system/authentication",
    "sid": 1729
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user",
    "sid": 1730
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/
      user-authentication-order",
    "sid": 1731
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key",
    "sid": 1732
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/algorithm",
    "sid": 1733
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/key-data",
    "sid": 1734
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/name",
    "sid": 1735
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      name",
    "sid": 1736
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      password",
```

```
    "sid": 1737
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock",
    "sid": 1738
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock/timezone-name",
    "sid": 1739
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock/timezone-utc-offset",
    "sid": 1740
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/contact",
    "sid": 1741
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver",
    "sid": 1742
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options",
    "sid": 1743
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options/
      attempts",
    "sid": 1744
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options/
      timeout",
    "sid": 1745
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/search",
    "sid": 1746
  }
```

```
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/server",
  "sid": 1747
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/server/name",
  "sid": 1748
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/server/
    udp-and-tcp",
  "sid": 1749
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/server/
    udp-and-tcp/address",
  "sid": 1750
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/dns-resolver/server/
    udp-and-tcp/port",
  "sid": 1751
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/hostname",
  "sid": 1752
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/location",
  "sid": 1753
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp",
  "sid": 1754
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/enabled",
  "sid": 1755
}
```

```
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server",
      "sid": 1756
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/
        association-type",
      "sid": 1757
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/iburst",
      "sid": 1758
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/name",
      "sid": 1759
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/prefer",
      "sid": 1760
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp",
      "sid": 1761
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp/address",
      "sid": 1762
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp/port",
      "sid": 1763
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius",
      "sid": 1764
    },
    {

```

```
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options",
    "sid": 1765
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options/attempts",
    "sid": 1766
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options/timeout",
    "sid": 1767
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server",
    "sid": 1768
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/
      authentication-type",
    "sid": 1769
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/name",
    "sid": 1770
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp",
    "sid": 1771
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
      address",
    "sid": 1772
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
      authentication-port",
    "sid": 1773
  },
  {

```

```
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
                  shared-secret",
    "sid": 1774
  }
]
```

Appendix B. SID auto generation

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.
2. The list of items is sorted in alphabetical order, 'namespace' in descending order, 'identifier' in ascending order. The 'namespace' and 'identifier' formats are described in the YANG module 'ietf-sid-file' defined in Section 4.
3. SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta between a reference SID and the current SID is used by protocols aiming to reduce message size.
4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

Changes of SID files can also be automated using the same method described above, only unassigned YANG items are processed at step #3. Already existing items in the SID file should not be given new SIDs.

Appendix C. ".sid" file lifecycle

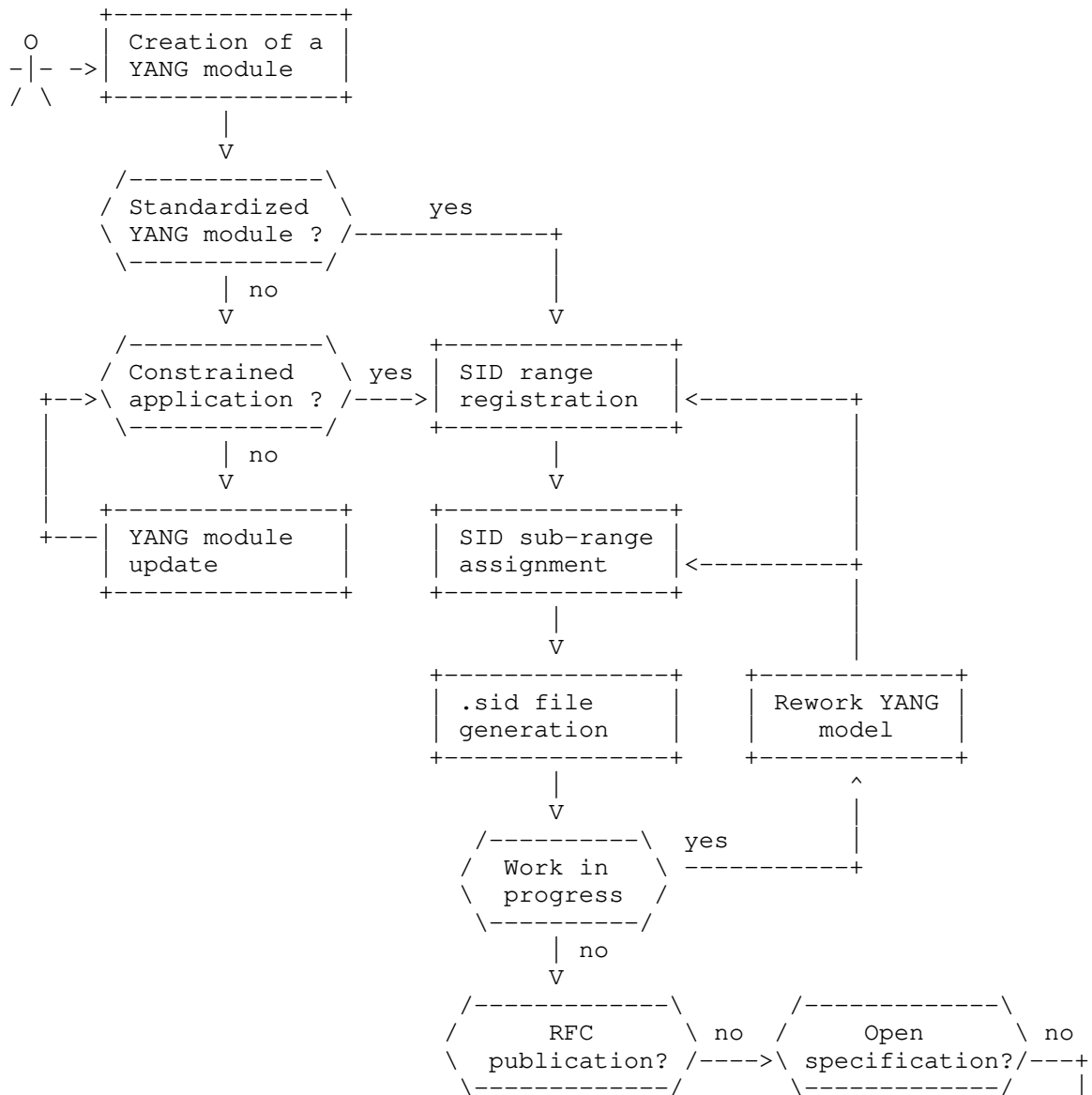
Before assigning SIDs to their YANG modules, YANG module authors must acquire a SID range from a "SID Range Registry". If the YANG module is part of an IETF draft or RFC, the SID range need to be acquired from the "IETF SID Range Registry" as defined in Section 6.3. For the other YANG modules, the authors can acquire a SID range from any "SID Range Registry" of their choice.

Once the SID range is acquired, the owner can use it to generate ".sid" file/s for his YANG module/s. It is recommended to leave some unallocated SIDs following the allocated range in each ".sid" file in order to allow better evolution of the YANG module in the future. Generation of ".sid" files should be performed using an automated

tool. Note that ".sid" files can only be generated for YANG modules and not for submodules.

C.1. SID File Creation

The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



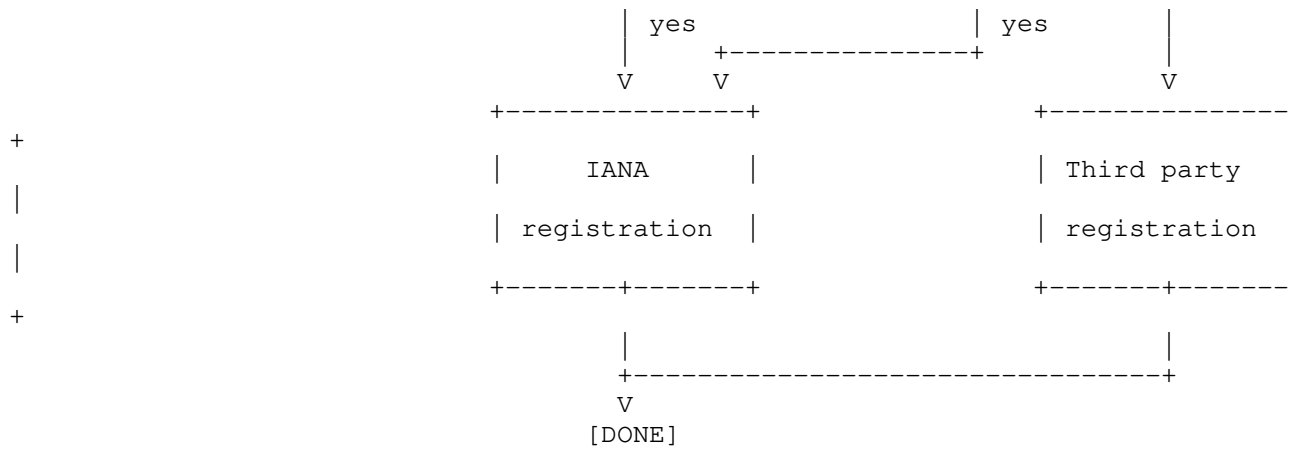


Figure 1: SID Lifecycle

C.2. SID File Update

The following Activity diagram summarizes the update of a YANG module and its associated .sid file.

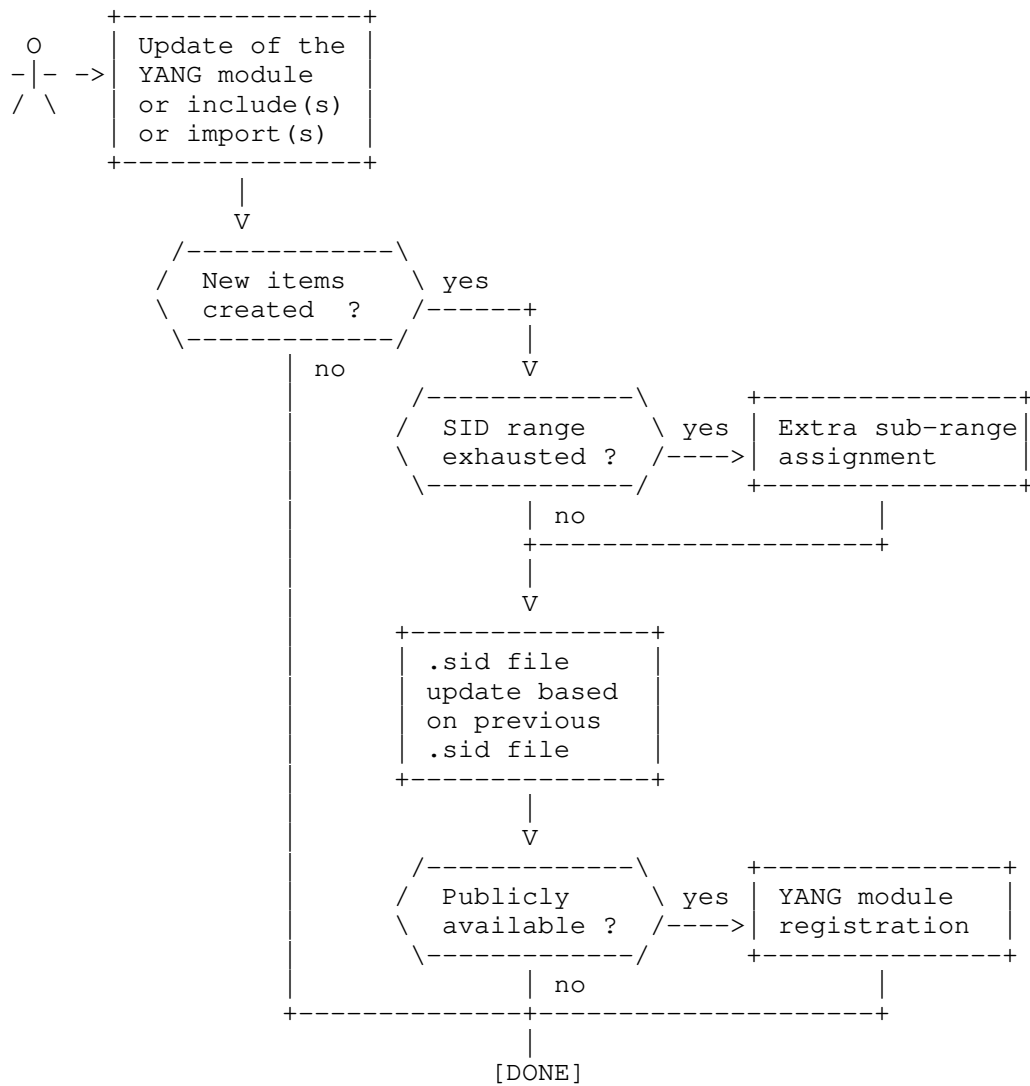


Figure 2: YANG and SID file update

Authors' Addresses

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

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

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

Email: a@ackl.io

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

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Updates: 7252, 8323 (if approved)
Intended status: Standards Track
Expires: May 1, 2020

K. Hartke
Ericsson
October 29, 2019

Extended Tokens and Stateless Clients
in the Constrained Application Protocol (CoAP)
draft-ietf-core-stateless-03

Abstract

This document provides considerations for alleviating CoAP clients and intermediaries of keeping per-request state. To facilitate this, this document additionally introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323.

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 May 1, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Extended Tokens	4
2.1. Extended Token Length (TKL) Field	4
2.2. Discovering Support	5
2.2.1. Extended-Token-Length Capability Option	5
2.2.2. Trial-and-Error	6
2.3. Intermediaries	8
3. Stateless Clients	8
3.1. Serializing Client State	8
3.2. Stateless Intermediaries	9
3.3. Extended Tokens	10
3.4. Stateless Message Transmission	11
4. Security Considerations	12
4.1. Extended Tokens	12
4.2. Stateless Clients	12
5. IANA Considerations	13
5.1. CoAP Signaling Option Number	13
6. References	13
6.1. Normative References	13
6.2. Informative References	14
Appendix A. Updated Message Formats	14
A.1. CoAP over UDP	15
A.2. CoAP over TCP	16
A.3. CoAP over WebSockets	17
Acknowledgements	18
Author's Address	18

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a RESTful application-layer protocol for constrained environments [RFC7228]. In CoAP, clients (or intermediaries in the client role) make requests to servers (or intermediaries in the server role), which satisfy the requests by returning responses.

While a request is ongoing, a client typically needs to keep some state that it requires for processing the response when it arrives. Identification of this state is done by means of a `_token_` in CoAP, an opaque sequence of bytes chosen by the client and included in the CoAP request, which is returned by the server verbatim in any resulting CoAP response (Figure 1).

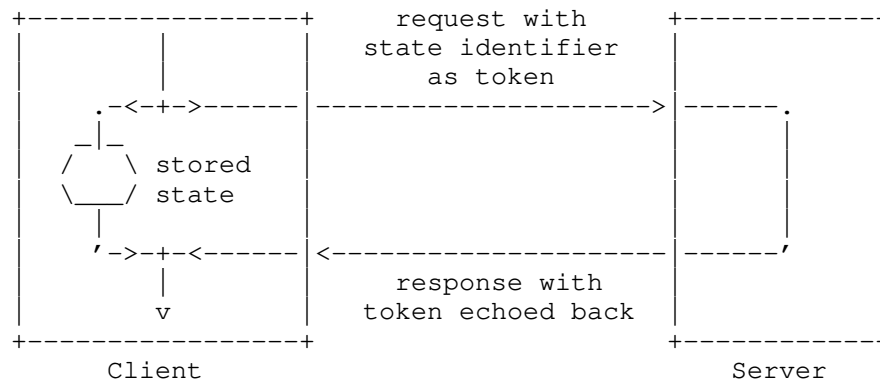


Figure 1: Token as an Identifier for Request State

In some scenarios, it can be beneficial to reduce the amount of state that is stored at the client at the cost of increased message sizes. A client can opt into this by serializing (parts of) its state into the token itself and then recovering this state from the token in the response (Figure 2).

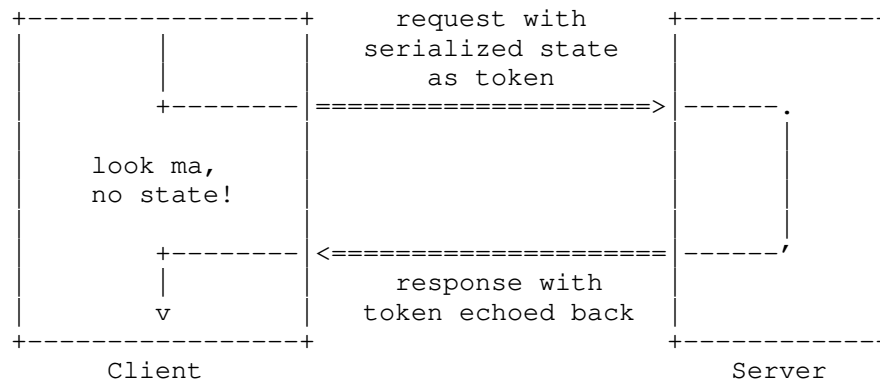


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for clients opting into becoming "stateless" in this way. (As those considerations will show, the term "stateless" is not entirely accurate. The clients still need to maintain per-server state and other kinds of state. So it is more accurate to say the these clients are just avoiding per-request state.)

Serializing state into tokens is complicated by the fact that both CoAP over UDP [RFC7252] and CoAP over reliable transports [RFC8323] limit the maximum token length to 8 bytes. To overcome this

limitation, Section 2 of this document first introduces a CoAP protocol extension for extended token lengths.

While the use case (avoiding per-request state) and the mechanism (extended token lengths) presented in this document are closely related, both can be used independently of each other: Some implementations may be able to fit their state in just 8 bytes; some implementations may have other use cases for extended token lengths.

1.1. Terminology

In this document, the term "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that does not require it to keep state for the individual requests it sends to a server (or intermediary in the server role). The client still needs to keep state for each server it communicates with (e.g., for token generation, message retransmission, and congestion control).

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. Extended Tokens

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with a new definition of the TKL field.

2.1. Extended Token Length (TKL) Field

The definition of the TKL field is updated as follows:

Token Length (TKL): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the variable-length Token field in bytes. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer precedes the Token field and indicates the length of the Token field minus 13.
- 14: A 16-bit unsigned integer in network byte order precedes the Token field and indicates the length of the Token field minus 269.
- 15: Reserved. This value MUST NOT be sent and MUST be processed as a message format error.

This increases the maximum token length that can be represented in a message to 65804 bytes. The maximum token length that sender and recipient implementations support may be shorter. For example, a constrained node of Class 1 [RFC7228] might support extended token lengths only up to 32 bytes.

All other fields retain their definition.

The updated message formats are illustrated in Appendix A.

2.2. Discovering Support

Extended token lengths require support from the server. Support can be discovered by a client in one of two ways:

- o Where Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Length Capability Option defined in Section 2.2.1.
- o Otherwise, such as in CoAP over UDP, support can only be discovered by trial-and-error, as described in Section 2.2.2.

2.2.1. Extended-Token-Length Capability Option

A server can use the elective Extended-Token-Length Capability Option to indicate the maximum token length it can accept in requests.

#	C	R	Applie s to	Name	Forma t	Length	Base Value
TB D			CSM	Extended-Token- Length	uint	0-3	8

C=Critical, R=Repeatable

Table 1: The Extended-Token-Length Capability Option

As per Section 3 of RFC 7252, the base value (and the value used when this option is not implemented) is 8.

The active value of the Extended-Token-Length Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

The option value MUST NOT be less than 8 or greater than 65804. If an option value outside this range is received, the value MUST be clamped to this range.

Any option value greater than 8 implies support for the new definition of the TKL field specified in Section 2.1. Indication of support by a server does not oblige a client to actually make use of token lengths greater than 8.

If a server receives a request with a token of a length greater than it indicated in its Extended-Token-Length Option, it MUST handle the request as a message format error.

Note: The Extended-Token-Length Capability Option does not apply to responses. The sender of a request is simply expected not to send a token of a length greater than it is willing to accept in a response.

2.2.2. Trial-and-Error

A server that does not support the updated definition of the TKL field specified in Section 2.1 will consider a request with a TKL field value outside the range 0 to 8 a message format error and rejected it (Section 3 of RFC 7252). A client can therefore determine support by sending a request with an extended token length and checking whether it's rejected by the server or not.

In CoAP over UDP, a Confirmable message with a message format error is rejected with a Reset message (Section 4.2 of RFC 7252). A Non-confirmable message with a message format error may be rejected with a Reset message or just silently ignored (Section 4.3 of RFC 7252). It is therefore RECOMMENDED that clients use a Confirmable message for determining support.

As per RFC 7252, Reset messages are empty and do not contain a token; they only return the Message ID (Figure 3). They also do not contain any indication of what caused a message format error. To avoid any ambiguity, it is therefore RECOMMENDED that clients use a request that has no potential message format error other than the extended token length.

An example of a suitable request is a Confirmable GET request that includes an If-None-Match option and a token of the greatest length that the client intends to use. Any response with the same token echoed back indicates that tokens up to that length are supported by the server.

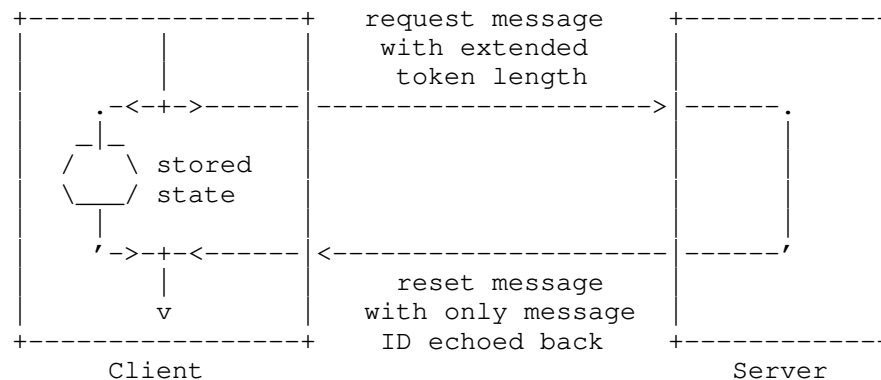


Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Server Does Not Have Support

A client SHOULD NOT assume that extended token lengths are supported by a server 60 minutes after receiving a response with an extended token length, as network addresses may change.

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to handle, it MUST NOT reject the message, as that would imply that extended token lengths are not supported at all. Instead, if the server cannot handle the request at the time, it SHOULD return a 5.03 (Service Unavailable) response; if the server will never be able to handle (e.g., because the token is too large), it SHOULD return a 4.00 (Bad Request) response.

Design Note: The requirement to return an error response when a request cannot be handled might seem somewhat contradictory. However, handling a request usually involves a number of steps from receiving the message to handing it over to application logic. The idea is that a server implementing this document at least should support large tokens in the first few steps, enough to return an error response rather than a Reset message.

Design Note: For simplicity, no mechanism to indicate the maximum supported token length is defined: A client implementation would probably already choose the shortest token possible for the task (such as being stateless, as described in Section 3), so it probably would not be able to reduce the length any further anyway, should a server indicate a lower limit.

2.3. Intermediaries

Tokens are a hop-by-hop feature: If there are one or more intermediaries between a client and a server, every token is scoped to the exchange between a node in the client role and the node in the server role that it is immediately interacting with.

When an intermediary receives a request, the only requirement is that it echoes the token back in any resulting response. There is no requirement or expectation that an intermediary passes a client's token on to a server or that an intermediary uses extended token lengths itself in a request to satisfy a request with an extended token length. Discovery needs to be performed for each hop.

3. Stateless Clients

A client can be alleviated of keeping per-request state by serializing the state into a sequence of bytes and sending the bytes as the token of the request. The server returns the token verbatim in the response to the client, which allows the client to recover the state and process the response as if it had kept the state locally.

The format of the serialized state is generally an implementation detail of the client and opaque to any server. However, transporting client state in requests and responses has significant security and privacy implications that need to be taken into consideration by a client implementation.

Furthermore, there are several non-obvious implications from CoAP protocol features that should be taken into consideration by a client implementation.

The following subsections discuss some of these considerations.

3.1. Serializing Client State

Serialized state information is an attractive target for both unwanted nodes (attackers between the node in client role and the node in server role) and wanted nodes (the node in server role itself) on the path. Therefore, a node in the client role SHOULD integrity protect the state information, unless processing a response does not modify state or cause any other significant side effects.

Even when the serialized state is integrity protected, an attacker may still replay a response, making the client believe it sent the same request twice. Therefore, the node in client role SHOULD implement replay protection (e.g., by using sequence numbers and a replay window), unless processing a response does not modify state or

cause other any significant side effects. Integrity protection is REQUIRED for replay protection.

If processing a response without keeping request state is sensitive to the time elapsed to sending the request, then the serialized state SHOULD include freshness information (e.g., a timestamp).

Information in the serialized state may be privacy sensitive. A node in client role SHOULD encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise. For example, an intermediary that serializes client information into its token leaks that information to the node in server role, which may be undesirable.

3.2. Stateless Intermediaries

Tokens are a hop-by-hop feature: If a client makes a request to an intermediary, that intermediary needs to store the client's token (along with the client's transport address) while it makes its own request towards the origin server and waits for the response. When the intermediary receives the response, it looks up the client's token and transport address for the received request and sends an appropriate response to the client.

Such an intermediary might want to be "stateless" as well, i.e., be alleviated of storing the client's token and transport address for received requests. This can be implemented by serializing this information along the request state into the token towards the origin server. When the intermediary receives the response, it can recover the information from the token and use it to satisfy the client's request.

The drawback of this approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which can significantly reduce efficiency.

In particular, when multiple clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of RFC 7641). This requirement cannot be satisfied without keeping request state; therefore, an intermediary MUST NOT include an Observe Option in requests it sends without keeping request state.

When using block-wise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. To ensure that this does not lead to inconsistent resource state, a stateless intermediary MUST include the Request-Tag Option [I-D.ietf-core-echo-request-tag] in

block-wise transfers with a value that uniquely identifies the client in the intermediary's namespace.

3.3. Extended Tokens

A client (or intermediary in the role of a client) that depends on support for extended token lengths (Section 2) from the server (or intermediary in the role of a server) to avoid keeping request state SHOULD perform a discovery of support (Section 2.2) before it can be stateless.

This discovery MUST be performed in a stateful way, i.e., keeping state for the request (Figure 4): If the client was stateless from the start and the server does not support extended tokens, then any error message could not be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

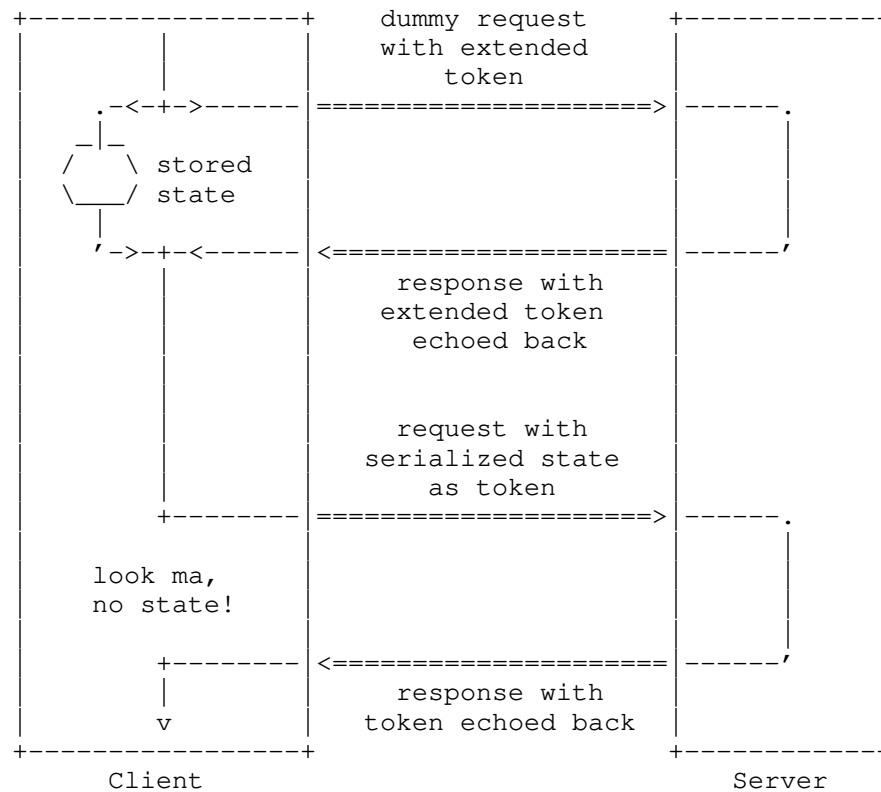


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

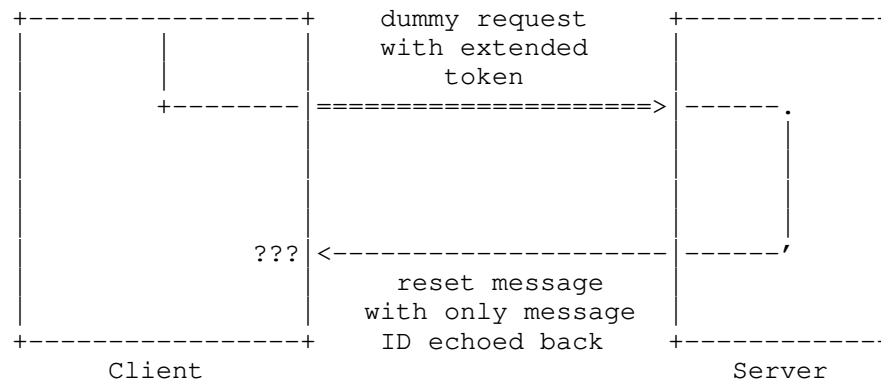


Figure 5: Stateless Discovery of Support Does Not Work

In environments where support can be reliably discovered through some other means, the discovery of support is OPTIONAL. An example for this is the Constrained Join Protocol (CoJP) in a 6TiSCH network [I-D.ietf-6tisch-minimal-security], where support for extended tokens is required from all relevant parties.

3.4. Stateless Message Transmission

As a further step, a client (or intermediary in the client role) might want to also avoid keeping message transmission state when using CoAP over UDP [RFC7252].

Generally, a client can use Confirmable or Non-confirmable messages for requests. When using Confirmable messages, it needs to keep message exchange state for performing retransmissions and handling Acknowledgement and Reset messages. When using Non-confirmable messages, it can keep no message exchange state. (However, in either case the client needs to keep congestion control state. That is, it needs to maintain state for each node it communicates with and, e.g., enforce NSTART.)

As per RFC 7252, a client must be prepared to receive a response as a piggybacked response, a separate response or Non-confirmable response (Section 5.2 of RFC 7252), regardless of the message type used for the request. A stateless client MUST handle these response types as follows:

- o If a piggybacked response passes the token integrity protection and freshness checks, the client processes the message as specified in RFC 7252; otherwise, it silently discards the message.

- o If a separate response passes the token integrity protection and freshness checks, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.2 of RFC 7252.
- o If a Non-confirmable response passes the token integrity protection and freshness checks, the client processes the message as specified in RFC 7252; otherwise, it rejects the message as specified in Section 4.3 of RFC 7252.

4. Security Considerations

4.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications in particular for nodes with constrained memory size that need to be mitigated. A node in the server role supporting extended token lengths may be vulnerable to a denial-of-service when an attacker (either on-path or a malicious client) sends large tokens to fill up the memory of the node. Implementations should be prepared to handle such messages.

4.2. Stateless Clients

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and non-obvious security and privacy implications that need to be mitigated; see Section 3.1.

The use of encryption, integrity protection, and replay protection of serialized state is recommended in general, unless a careful analysis of any potential attacks to security and privacy is performed. AES-CCM with a 64 bit tag is recommended, combined with a sequence number and a replay window. Where encryption is not needed, HMAC-SHA-256, combined with a sequence number and a replay window, may be used.

When using AES-CCM, repeated use of the same nonce under the same key causes the cipher to fail catastrophically. If a nonce is ever used for more than one encryption operation with the same key, then the same key stream gets used to encrypt both plaintexts and the confidentiality guarantees are voided. Devices with low-entropy sources -- as is typical with constrained devices, which incidentally happen to be a natural candidate for the stateless mechanism described in this document -- need to carefully pick a nonce generation mechanism that provides the above uniqueness guarantee. Additionally, since it can be difficult to use AES-CCM securely when using statically configured keys, implementations should use automated key management [RFC4107].

5. IANA Considerations

5.1. CoAP Signaling Option Number

The following entries are added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

Applies to	Number	Name	Reference
7.01	TBD	Extended-Token-Length	[[this document]]

6. References

6.1. Normative References

- [I-D.ietf-core-echo-request-tag]
 Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-07 (work in progress), September 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <<https://www.rfc-editor.org/info/rfc4107>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [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/info/rfc7959>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.

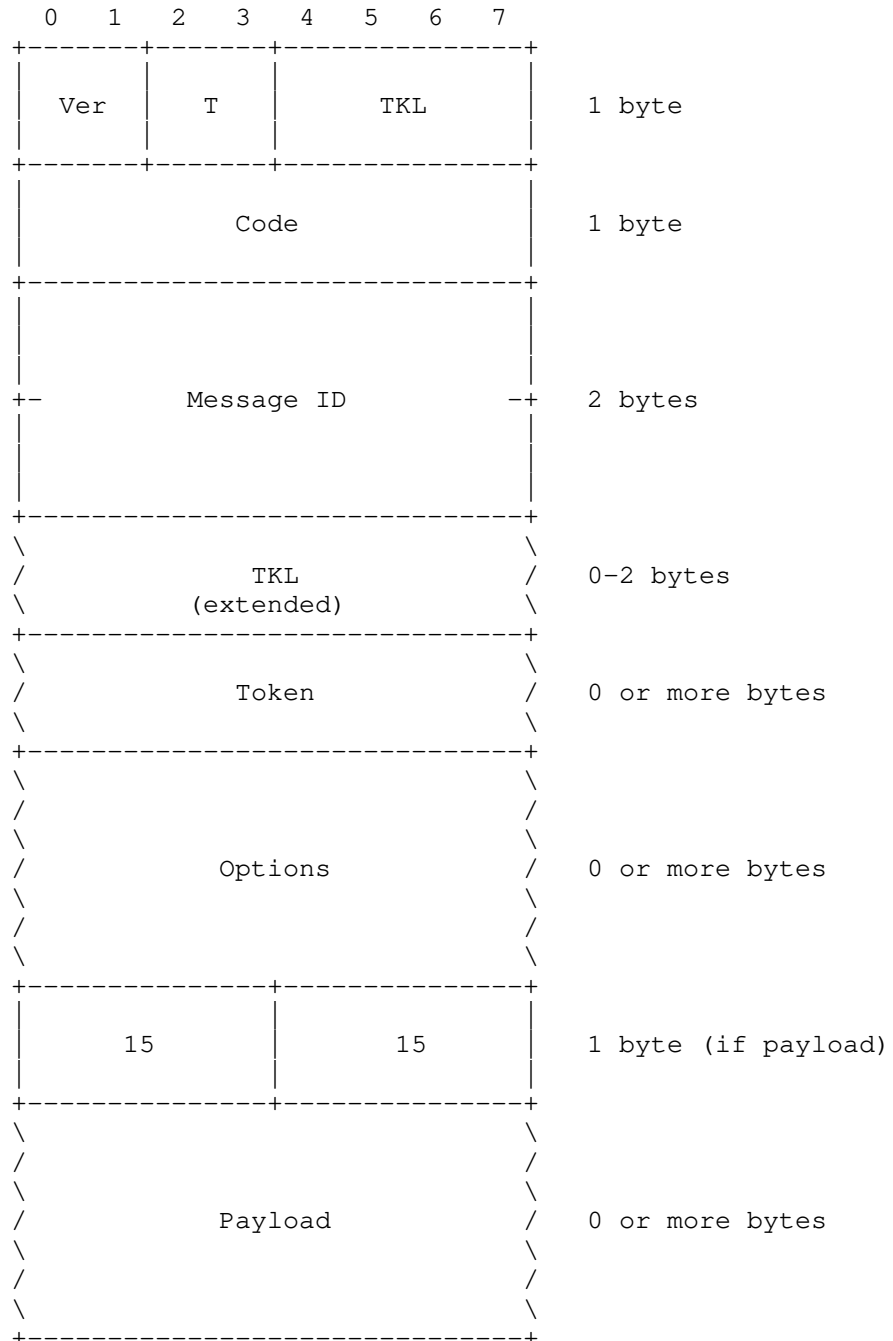
6.2. Informative References

- [I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson,
"Minimal Security Framework for 6TiSCH", draft-ietf-
6tisch-minimal-security-13 (work in progress), October
2019.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained-Node Networks", RFC 7228,
DOI 10.17487/RFC7228, May 2014,
<<https://www.rfc-editor.org/info/rfc7228>>.

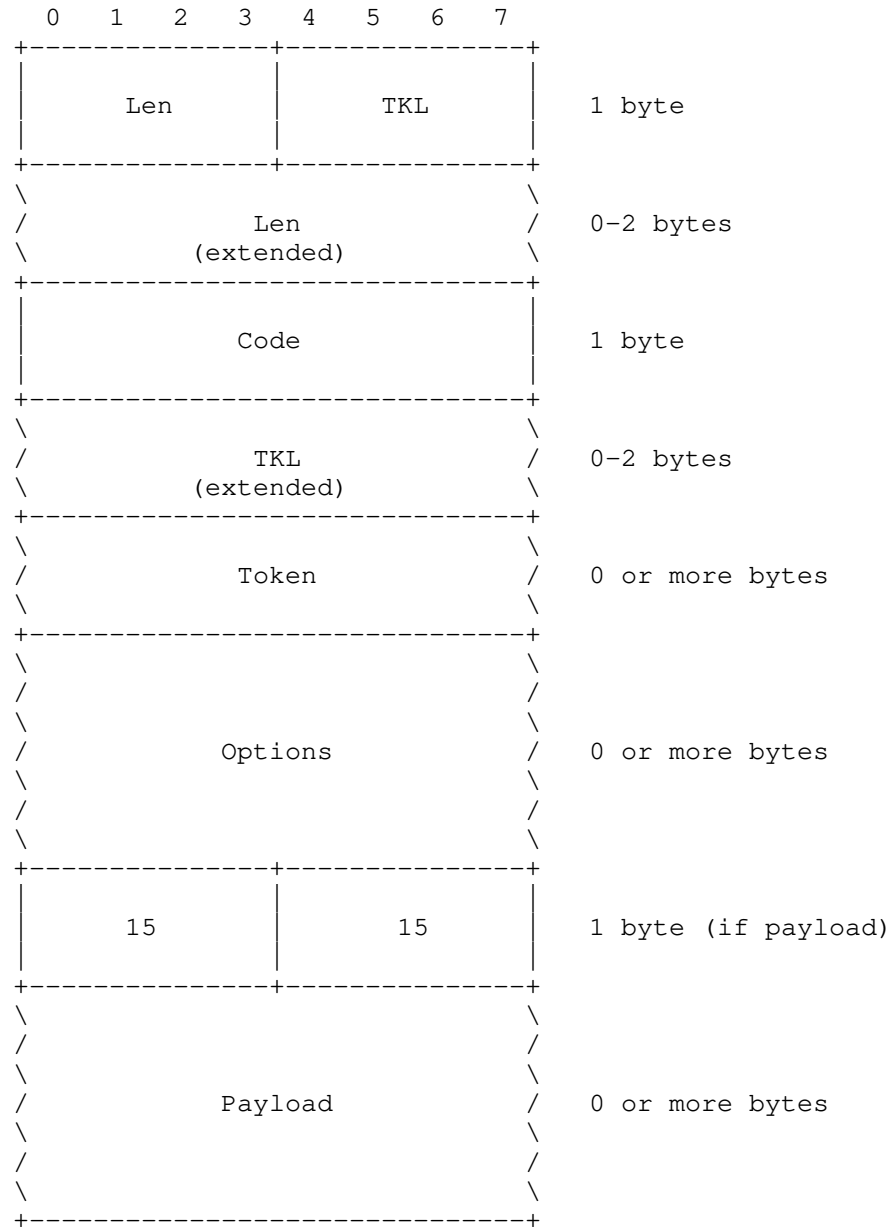
Appendix A. Updated Message Formats

This appendix illustrates the CoAP message formats updated with the new definition of the TKL field (Section 2).

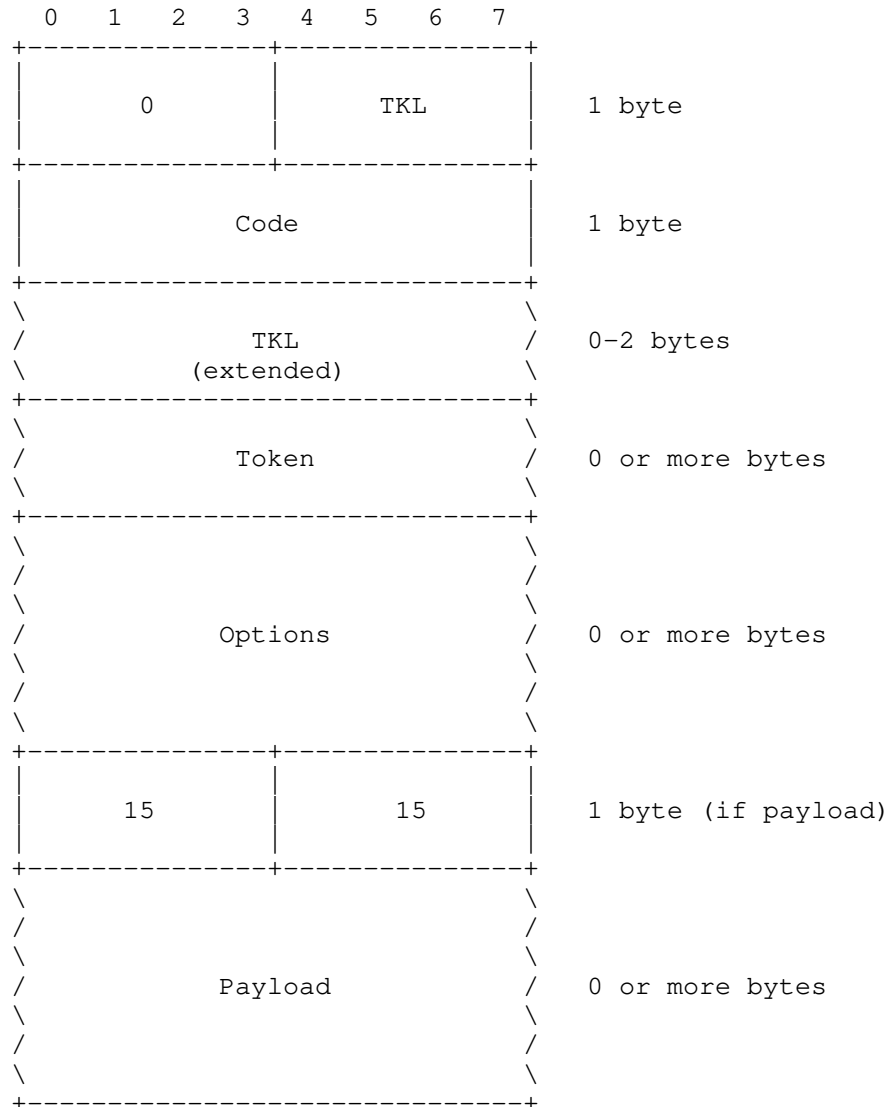
A.1. CoAP over UDP



A.2. CoAP over TCP



A.3. CoAP over WebSockets



Acknowledgements

This document is based on the requirements of and work on the Minimal Security Framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] by Malisa Vucinic, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Christian Amsuess, Carsten Bormann, Ari Keranen, John Mattsson, Jim Schaad, Goeran Selander, and Malisa Vucinic for helpful comments and discussions that have shaped the document.

Special thanks to Thomas Fossati for his in-depth review, copious comments, and suggested text.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: March 12, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
A. Pelov
Acklio
September 09, 2019

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-11

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 12, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	3
3. Properties of the CBOR Encoding	5
3.1. CBOR diagnostic notation	5
3.2. YANG Schema Item iDentifier (SID)	6
3.3. Name	7
4. Encoding of YANG Schema Node Instances	9
4.1. The 'leaf'	9
4.1.1. Using SIDs in keys	9
4.1.2. Using names in keys	9
4.2. The 'container' and other collections	10
4.2.1. Using SIDs in keys	11
4.2.2. Using names in keys	12
4.3. The 'leaf-list'	13
4.3.1. Using SIDs in keys	14
4.3.2. Using names in keys	14
4.4. The 'list' and 'list' instance(s)	15
4.4.1. Using SIDs in keys	16
4.4.2. Using names in keys	18
4.5. The 'anydata'	20
4.5.1. Using SIDs in keys	21
4.5.2. Using names in keys	22
4.6. The 'anyxml'	23
4.6.1. Using SIDs in keys	23
4.6.2. Using names in keys	24
5. Encoding of YANG data templates	24
5.1. Using SIDs in keys	25
5.2. Using names in keys	26
6. Representing YANG Data Types in CBOR	27
6.1. The unsigned integer Types	27
6.2. The integer Types	28
6.3. The 'decimal64' Type	28
6.4. The 'string' Type	29
6.5. The 'boolean' Type	29
6.6. The 'enumeration' Type	29
6.7. The 'bits' Type	30
6.8. The 'binary' Type	32
6.9. The 'leafref' Type	32
6.10. The 'identityref' Type	33
6.10.1. SIDs as identityref	33
6.10.2. Name as identityref	34
6.11. The 'empty' Type	34
6.12. The 'union' Type	35

6.13. The 'instance-identifier' Type	36
6.13.1. SIDs as instance-identifier	36
6.13.2. Names as instance-identifier	39
7. Security Considerations	41
8. IANA Considerations	41
8.1. CBOR Tags Registry	41
9. Acknowledgments	41
10. References	42
10.1. Normative References	42
10.2. Informative References	42
Authors' Addresses	43

1. Introduction

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

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

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

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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.

The following terms are defined in [RFC7950]:

- o action
- o anydata
- o anyxml
- o data node

- o data tree
- o datastore
- o feature
- o identity
- o module
- o notification
- o RPC
- o schema node
- o schema tree
- o submodule

The following terms are defined in [RFC8040]:

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

This specification also makes use of the following terminology:

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

3. Properties of the CBOR Encoding

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

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 3.2 and names as defined in Section 3.3. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of SID not encoded as delta, anyxml schema nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

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

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

Application payloads carrying a value serialized using the rules defined by this specification (e.g. CoAP Content-Format) SHOULD include the identifier (e.g. SID, namespace qualified name, instance-identifier) of this value. When SIDs are used as identifiers, the reference SID SHALL be included in the payload to allow stateless conversion of delta values to SIDs.

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

3.1. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for

documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

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

Table 1: CBOR diagnostic notation summary

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

3.2. YANG Schema Item iDentifier (SID)

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

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

To minimize their size, SIDs used as keys in inner CBOR maps are typically encoded using deltas. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized. These SIDs may also be encoded as absolute number when enclosed by CBOR tag 47.

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

3.3. Name

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

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

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

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

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

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

Figure 1: ABNF Production for a simple or namespace qualified name

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

Definition example:

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

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

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

CBOR diagnostic notation:

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

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

4. Encoding of YANG Schema Node Instances

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

4.1. The 'leaf'

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

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

Definition example from [RFC7317]:

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

4.1.1. Using SIDs in keys

CBOR diagnostic notation:

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

CBOR encoding:

```
A1                                # map(1)  
  19 06D8                        # unsigned(1752)  
  72                             # text(18)  
    6D796866F73742E6578616D706C652E636F6D # "myhost.example.com"
```

4.1.2. Using names in keys

CBOR diagnostic notation:

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

CBOR encoding:

```

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

```

4.2. The 'container' and other collections

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

This specification supports two type of CBOR keys; SID as defined in Section 3.2 and names as defined in Section 3.3.

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

Definition example from [RFC7317]:

```

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

container system-state {

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

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

```

4.2.1. Using SIDs in keys

In the context of containers and other collections, CBOR map keys within inner CBOR maps can be encoded using deltas or SIDs. In the case of deltas, they MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual delta value. In the case of SID, they are encoded using the SID value enclosed by CBOR tag 47 as defined in Section 8.1.

Delta values are computed as follows:

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

CBOR diagnostic notation:

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

CBOR encoding:

```

A1                                # map(1)
  19 06B8                        # unsigned(1720)
  A1                              # map(1)
    01                          # unsigned(1)
    A2                          # map(2)
      02                        # unsigned(2)
      78 1A                    # text(26)
        323031352D31302D30325431343A34373A32345A2D30353A3030
      01                        # unsigned(1)
      78 1A                    # text(26)
        323031352D30392D31355430393A31323A35385A2D30353A3030

```

Figure 2: System state clock encoding

4.2.2. Using names in keys

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

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

Definition example from [RFC7317]:

```

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

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

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

```

CBOR diagnostic notation:


```

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

```

CBOR encoding:

```

A1                                     # map(1)
  78 18                               # text(24)
    6965746662D73797374656D3A73797374656D2D7374617465
  A1                                   # map(1)
    65                               # text(5)
      636C6F636B                     # "clock"
    A2                               # map(2)
      70                             # text(16)
        63757272656E742D6461746574696D65
      78 1A                           # text(26)
        323031352D31302D30325431343A34373A32345A2D30353A3030
      6D                             # text(13)
        626F6F742D6461746574696D65
      78 1A                           # text(26)
        323031352D30392D31355430393A31323A35385A2D30353A3030

```

4.3. The 'leaf-list'

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

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

Definition example [RFC7317]:

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

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

4.3.1. Using SIDs in keys

CBOR diagnostic notation:

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

CBOR encoding:

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

4.3.2. Using names in keys

CBOR diagnostic notation:

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

CBOR encoding:


```

list server {
  key name;

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

```

4.4.1. Using SIDs in keys

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

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

A1                                     # map(1)
  19 06DC                             # unsigned(1756)
  82                                   # array(2)
    A5                               # map(5)
      03                             # unsigned(3)
      6E                             # text(14)
        4E52432054494320736572766572 # "NRC TIC server"
      05                             # unsigned(5)
      A2                             # map(2)
        01                           # unsigned(1)
        6A                           # text(10)
          74696332E6E72632E6361      # "tic.nrc.ca"
        02                           # unsigned(2)
        18 7B                        # unsigned(123)
      01                             # unsigned(1)
      00                             # unsigned(0)
      02                             # unsigned(2)
      F4                             # primitive(20)
      04                             # unsigned(4)
      F5                             # primitive(21)
    A2                               # map(2)
      03                             # unsigned(3)
      6E                             # text(14)
        4E52432054414320736572766572 # "NRC TAC server"
      05                             # unsigned(5)
      A1                             # map(1)
        01                           # unsigned(1)
        6A                           # text(10)
          74616332E6E72632E6361      # "tac.nrc.ca"

```

4.4.2. Using names in keys

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

CBOR diagnostic notation:

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

CBOR encoding:

```

A1                                     # map(1)
  72                                 # text(18)
    696574662D73797374656D3A736572766572
  82                                 # array(2)
    A5                             # map(5)
      64                           # text(4)
        6E616D65                  # "name"
      6E                           # text(14)
        4E52432054494320736572766572
      63                           # text(3)
        756470                    # "udp"
      A2                           # map(2)
        67                        # text(7)
          61646472657373          # "address"
        6A                        # text(10)
          7469632E6E72632E6361    # "tic.nrc.ca"
        64                        # text(4)
          706F7274                # "port"
        18 7B                    # unsigned(123)
      70                          # text(16)
        6173736F63696174696F6E2D74797065
      00                          # unsigned(0)
      66                          # text(6)
        696275727374             # "iburst"
      F4                          # primitive(20)
      66                          # text(6)
        707265666572             # "prefer"
      F5                          # primitive(21)
    A2                            # map(2)
      64                          # text(4)
        6E616D65                  # "name"
      6E                          # text(14)
        4E52432054414320736572766572
      63                          # text(3)
        756470                    # "udp"
      A1                          # map(1)
        67                        # text(7)
          61646472657373          # "address"
        6A                        # text(10)
          7461632E6E72632E6361    # "tac.nrc.ca"

```

4.5. The 'anydata'

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

- o CBOR map keys of any inner schema nodes MUST be set to valid deltas or names.
- o The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).
- o CBOR map values MUST follow the encoding rules of one of the datatypes listed in Section 4.

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

Definition example:

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

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

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

4.5.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  60123 : {
    77 : {
      1 : "0/4/21",
      2 : "Open pin 2"
    }
  }
}
```

/ last-event (SID 60123) /
/ event (SID 60200) /
/ port-name (SID 60201) /
/ port-fault (SID 60202) /

CBOR encoding:

```

A1                                # map(1)
  19 EADB                        # unsigned(60123)
  A1                              # map(1)
    18 4D                        # unsigned(77)
    A2                          # map(2)
      18 4E                      # unsigned(78)
      66                        # text(6)
        302F342F3231           # "0/4/21"
      18 4F                      # unsigned(79)
      6A                        # text(10)
        4F70656E2070696E2032  # "Open pin 2"

```

In some implementations, it might be simpler to use the absolute SID tag encoding for the anydata root element. The resulting encoding is as follow:

```

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

```

4.5.2. Using names in keys

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

A1                                     # map(1)
  74                                 # text(20)
    6576656E742D6C6F673A6C6173742D6576656E74
A1                                     # map(1)
  78 20                             # text(32)
    6578616D706C652D706F72743A206578616D7
    06C652D706F72742D6661756C74
A2                                     # map(2)
  69                                 # text(9)
    706F72742D6E616D65             # "port-name"
  66                                 # text(6)
    302F342F3231                   # "0/4/21"
  6A                                 # text(10)
    706F72742D6661756C74           # "port-fault"
  6A                                 # text(10)
    4F70656E2070696E2032           # "Open pin 2"

```

4.6. The 'anyxml'

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

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

Definition example from [RFC7951]:

```

module bar-module {
  ...
  anyxml bar;
}

```

4.6.1. Using SIDs in keys

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

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

```

4.6.2. Using names in keys

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

A1          # map(1)
  6E        # text(14)
    6261722D6D6F647556C653A626172  # "bar-module:bar"
  83      # array(3)
    F5    # primitive(21)
    F6    # primitive(22)
    F5    # primitive(21)

```

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the 'yang-data' extension as described by [RFC8040].

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

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

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

```

import ietf-restconf {
  prefix rc;
}

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

```

5.1. Using SIDs in keys

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

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using SIDs as defined in Section 3.2.

CBOR diagnostic notation:

```

{
  1024 : {
    4 : 1011,
    1 : 1018,
    2 : 1740,
    3 : "Maximum exceeded"
  }
}

```

/ error (SID 1024) /
/ error-tag (SID 1028) /
/ = invalid-value (SID 1011) /
/ error-app-tag (SID 1025) /
/ = not-in-range (SID 1018) /
/ error-data-node (SID 1026) /
/ = timezone-utc-offset (SID 1740) /
/ error-message (SID 1027) /

CBOR encoding:

```

A1                                # map(1)
  19 0400                        # unsigned(1024)
  A4                              # map(4)
    04                          # unsigned(4)
    19 03F3                      # unsigned(1011)
    01                          # unsigned(1)
    19 03FA                      # unsigned(1018)
    02                          # unsigned(2)
    19 06CC                      # unsigned(1740)
    03                          # unsigned(3)
    70                          # text(16)
                                4D6178696D756D2065786365565646564

```

5.2. Using names in keys

YANG template encoded using names are carried in a CBOR map containing a single item pair. The key of this item is set to the namespace qualified name of the YANG template container, the value is set the CBOR encoding of this container as defined in Section 3.3.

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using names as defined Section 3.3.

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

A1                                     # map(1)
  6F                                 # text(15)
    696574662D636F6D693A6572726F72 # "ietf-comi:error"
  A4                                 # map(4)
    69                             # text(9)
      6572726F722D746167           # "error-tag"
    6D                             # text(13)
      696E76616C69642D76616C7565   # "invalid-value"
    6D                             # text(13)
      6572726F722D6170702D746167   # "error-app-tag"
    6C                             # text(12)
      6E6F742D696E2D72616E6765     # "not-in-range"
    6F                             # text(15)
      6572726F722D646174612D6E6F6465 # "error-data-node"
    73                             # text(19)
      74696D657A6F6E652D7574632D6F66666736574
                                     # "timezone-utc-offset"
    6D                             # text(13)
      6572726F722D6D657373616765   # "error-message"
    70                             # text(16)
      4D6178696D756D206578636565646564

```

6. Representing YANG Data Types in CBOR

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

6.1. The unsigned integer Types

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

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

Definition example from [RFC8344]:

```

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

```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: -300

CBOR encoding: 39 012B

6.3. The 'decimal64' Type

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

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

Definition example from [RFC7317]:

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

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

CBOR encoding: C4 82 21 19 0101

6.4. The 'string' Type

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

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

Definition example from [RFC8343]:

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

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: true

CBOR encoding: F5

6.6. The 'enumeration' Type

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: 3

CBOR encoding: 03

To avoid overlap of 'value' defined in different 'enumeration' statements, 'enumeration' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain one of the names assigned by 'enum' statements in YANG. The encoding MUST be enclosed by the enumeration CBOR tag as specified in Section 8.1.

Definition example from [RFC7950]:

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

CBOR diagnostic notation: 44("unbounded")

CBOR encoding: D8 2C 69 756E626F756E646564

6.7. The 'bits' Type

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

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are

assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of an 'alarm-state' leaf instance with the 'under-repair' and 'critical' flags set.

Definition example from [RFC8348]:

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

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

CBOR diagnostic notation: h'06'

CBOR encoding: 41 06

To avoid overlap of 'bit' defined in different 'bits' statements, 'bits' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain a space-separated sequence of names of 'bit' that are set. The encoding MUST be enclosed by the bits CBOR tag as specified in Section 8.1.

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

Definition example:

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

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

CBOR encoding: D8 2B 75 756E6465722D72657061697220637269746963616C

6.8. The 'binary' Type

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

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

Definition example:

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

CBOR diagnostic notation: h'1F1CE6A3F42660D888D92A4D8030476E'

CBOR encoding: 50 1F1CE6A3F42660D888D92A4D8030476E

6.9. The 'leafref' Type

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

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

Definition example from [RFC8343]:

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

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

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding `identityref`, a YANG Schema Item identifier (SID) as defined in Section 3.2 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as `identityref`

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

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

Definition example from [RFC7317]:

```

identity interface-type {
}

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

identity ethernetCsmacd {
  base iana-interface-type;
}

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

```

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

6.10.2. Name as identityref

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

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

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

CBOR encoding: 78 1b
69616E612D696662D747970653A65746865726E657443736D616364

6.11. The 'empty' Type

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

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

Definition example from [RFC8344]:

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

CBOR diagnostic notation: null

CBOR encoding: F6

6.12. The 'union' Type

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

- o bits
- o enumeration
- o identityref
- o instance-identifier

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

As mentioned in Section 6.6 and in Section 6.7, 'enumeration' and 'bits' are encoded as CBOR text string data item (major type 3) when defined within a 'union' type.

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

Definition example from [RFC7317]:

```

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

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

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

leaf address {
  type inet:ip-address;
}

```

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

CBOR encoding: 74 323030313A6462383A6130623A313266303A3A31

6.13. The 'instance-identifier' Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 3.2 and one based on names as defined in Section 3.3.

6.13.1. SIDs as instance-identifier

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

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

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

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

- o The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.
- o The following entries MUST contain the value of each key required to identify the instance of the targeted schema node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

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

Definition example from [RFC7950]:

```
container system {  
  ...  
  leaf reporting-entity {  
    type instance-identifier;  
  }  
  
  leaf contact { type string; }  
  
  leaf hostname { type inet:domain-name; } } ~~~~
```

First example:

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

Definition example from [RFC7317]:

```
container system {  
    leaf contact {  
        type string;  
    }  
  
    leaf hostname {  
        type inet:domain-name;  
    }  
}
```

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

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

Definition example from [RFC7317]:

```
list user {  
    key name;  
  
    leaf name {  
        type string;  
    }  
    leaf password {  
        type ianach:crypt-hash;  
    }  
  
    list authorized-key {  
        key name;  
  
        leaf name {  
            type string;  
        }  
        leaf algorithm {  
            type string;  
        }  
        leaf key-data {  
            type binary;  
        }  
    }  
}
```

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

CBOR encoding:

```

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

Third example:

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

CBOR diagnostic notation: [1730, "jack"]

CBOR encoding:

```

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

6.13.2. Names as instance-identifier

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

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

For example,

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

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

The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

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

CBOR encoding:

78 1c 2F696574662D73797374656D3A73797374656D2F636F6E74616374

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

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

CBOR encoding:

78 59
2F696574662D73797374656D3A73797374656D2F61757468656E74696361
74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A
65642D6B65790D0A5B6E616D653D2761646D696E275D2F6B65792D64617461

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

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

CBOR encoding:

78 33
2F696574662D73797374656D3A73797374656D2F61757468656E74696361
74696F6E2F757365725B6E616D653D27626F62275D

7. Security Considerations

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

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

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

8. IANA Considerations

8.1. CBOR Tags Registry

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

Tag	Data Item	Semantics	Reference
43	byte string	YANG bits datatype ; see Section 6.7.	[this]
44	unsigned integer	YANG enumeration datatype ; see Section 6.6.	[this]
45	unsigned integer or text string	YANG identityref datatype ; see Section 6.10.	[this]
46	unsigned integer or text string	YANG instance-identifier datatype; see Section 6.13.	[this] [this]
47	or array unsigned integer	YANG Schema Item iDentifier ; see Section 3.2.	[this]

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

9. Acknowledgments

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

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

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

10.2. Informative References

- [I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface", draft-ietf-core-comi-07 (work in progress), July 2019.
- [I-D.ietf-core-sid] Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.
- [RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <<https://www.rfc-editor.org/info/rfc8348>>.

Authors' Addresses

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

Email: michel.veillette@trilliantinc.com

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

Email: ivaylo@ackl.io

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

Email: a@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 25, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
Acklio
July 24, 2019

Constrained YANG Module Library
draft-ietf-core-yang-library-00

Abstract

This document describes a constrained version of the YANG library that provides information about the YANG modules, datastores, and datastore schemas used by a constrained network management server (e.g., a CORECONF server).

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 January 25, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology and Notation	2
3. Overview	3
3.1. Tree diagram	3
3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library	4
4. YANG Module "ietf-constrained-yang-library"	5
5. IANA Considerations	13
5.1. YANG Module Registry	13
6. Security Considerations	13
7. Acknowledgments	14
8. References	14
8.1. Normative References	14
8.2. Informative References	14
Authors' Addresses	15

1. Introduction

There is a need for a standard mechanism to expose which YANG modules, datastores and datastore schemas are in use by a constrained network management server. This document defines the YANG module 'ietf-constrained-yang-library' that provides this information.

YANG module 'ietf-constrained-yang-library' shares the same data model and objectives as 'ietf-yang-library', only datatypes and mandatory requirements have been updated to minimize its size to allow its implementation by Constrained Nodes and/or Constrained Networks as defined by [RFC7228]. To review the list of objectives and proposed data model, please refer to [RFC8525] section 2 and 3.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950]: client, deviation, feature, module, submodule and server.

The following term is defined in [I-D.ietf-core-sid]: YANG Schema Item Identifier (SID).

The following terms are defined in [RFC8525]: YANG library and YANG library checksum.

3. Overview

The conceptual model of the YANG library is depicted in Figure 1.

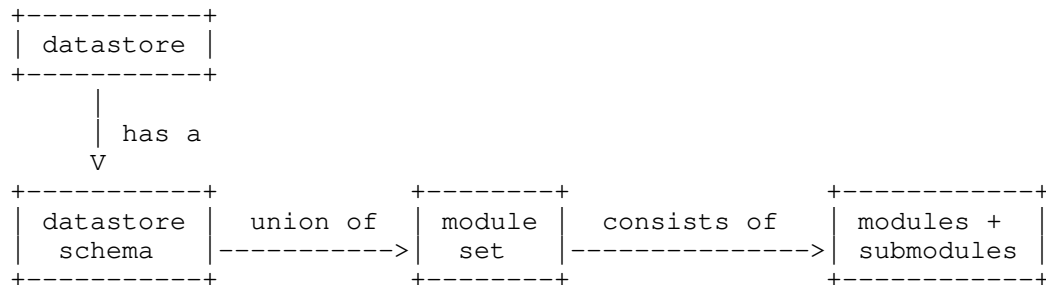


Figure 1: Conceptual model of the YANG library

It's expected that most constrained network management servers have one datastore (e.g. a unified datastore). However, some servers may have multiples datastore as described by NMDA [RFC8342]. The YANG library data model supports both cases.

In this model, every datastore has an associated datastore schema, which is the union of module sets, which is a collection of modules. Multiple datastores may refer to the same datastore schema and individual datastore schemas may share module sets.

For each module, the YANG library provides:

- o the YANG module identifier (i.e. SID)
- o its revision
- o its list of submodules
- o its list of imported modules
- o its set of features and deviations

YANG module namespace and location are also supported, but their implementation is not recommended for constrained servers.

3.1. Tree diagram

The tree diagram of YANG module `ietf-constrained-yang-library` is provided below. This graphical representation of a YANG module is defined in [RFC8340].

```

module: ietf-constrained-yang-library
  +--ro yang-library
    +--ro module-set* [index]
      +--ro index          uint8
      +--ro module* [identifier]
        +--ro identifier    sid:sid
        +--ro revision?     revision-identifier
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
        +--ro feature*      sid:sid
        +--ro deviation*    -> ../../module/identifier
      +--ro import-only-module* [identifier revision]
        +--ro identifier    sid:sid
        +--ro revision      union
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
    +--ro schema* [index]
      +--ro index          uint8
      +--ro module-set*    -> ../../module-set/index
    +--ro datastore* [identifier]
      +--ro identifier      ds:datastore-ref
      +--ro schema          -> ../../schema/index
    +--ro checksum          binary

notifications:
  +---n yang-library-update
    +--ro checksum -> /yang-library/checksum

```

3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library

The changes between the reference data model 'ietf-yang-library' and its constrained version 'ietf-constrained-yang-library' are listed below:

- o module-set 'name' and schema 'name' are implemented using an 8 bits unsigned integer and renamed 'index'.

- o module 'name', submodule 'name' and datastore 'name' are implemented using a SID (i.e. an unsigned integer) and renamed 'identifier'.
- o 'feature' and 'deviation' are implemented using a SID (i.e. an unsigned integer).
- o 'revision' fields are implemented using a 4 bytes binary string.
- o the mandatory requirement of the 'namespace' fields is removed, and implementation is not recommended. SIDs used by constrained devices and protocols don't require namespaces.
- o the implementation of the 'location' fields are not recommended, the use of the module SID as the handle to retrieve the associated YANG module is proposed instead.

4. YANG Module "ietf-constrained-yang-library"

RFC Ed.: update the date below with the date of RFC publication and remove this note.

```
<CODE BEGINS> file "ietf-constrained-yang-library@2019-03-28.yang"
module ietf-constrained-yang-library {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
  prefix "yanglib";

  // RFC Ed.: update ietf-core-sid reference.

  import ietf-sid-file {
    prefix sid;
    reference "I-D.ietf-core-sid";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA).";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
```

contact

"WG Web: <<http://datatracker.ietf.org/wg/core/>>
WG List: <<mailto:core@ietf.org>>
WG Chair: Carsten Bormann
<<mailto:cabo@tzi.org>>
WG Chair: Jaime Jimenez
<<mailto:jaime.jimenez@ericsson.com>>
Editor: Michel Veillette
<<mailto:michel.veillette@trilliantinc.com>>";

description

"This module provides information about the YANG modules, datastores, and datastore schemas implemented by a constrained network management server.

Copyright (c) 2018 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 (<http://trustee.ietf.org/license-info>).

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

// RFC Ed.: update reference.

```
revision 2019-03-28 {  
  description  
    "Second revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
revision 2018-09-21 {  
  description  
    "Initial revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
/*
 * Typedefs
 */

typedef revision-identifier {
  type binary {
    length "4";
  }
  description
    "Revision date encoded as a binary string, each nibble
    representing a digit of the of revision date. For example,
    revision 2018-09-21 is encoded as 0x20 0x18 0x09 0x21.";
}

/*
 * Groupings
 */

grouping module-identification-leafs {
  description
    "Parameters for identifying YANG modules and submodules.";

  leaf identifier {
    type sid:sid;
    mandatory true;
    description
      "SID assigned to this module or submodule.";
  }
  leaf revision {
    type revision-identifier;
    description
      "The YANG module or submodule revision date. If no
      revision statement is present in the YANG module
      or submodule, this leaf is not instantiated.";
  }
}

grouping location-leaf-list {
  description
    "Common location leaf list parameter for modules and
    submodules.";

  leaf-list location {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema resource
      for this module or submodule."
  }
}
```

```
        This leaf is present in the model to keep the alignment
        with 'ietf-yang-library'. Support of this leaf in
        constrained devices is not necessarily required, nor
        expected. It is recommended that clients used the module
        or sub-module SID as the handle used to retrieve the
        corresponding YANG module";
    }
}

grouping implementation-parameters {
  description
    "Parameters for describing the implementation of a module.";

  leaf-list feature {
    type sid:sid;
    description
      "List of all YANG feature names from this module that are
      supported by the server, regardless whether they are
      defined in the module or any included submodule.";
  }
  leaf-list deviation {
    type leafref {
      path "../..../module/identifier";
    }
    description
      "List of all YANG deviation modules used by this server to
      modify the conformance of the module associated with this
      entry. Note that the same module can be used for
      deviations for multiple modules, so the same entry MAY
      appear within multiple 'module' entries.

      This reference MUST NOT (directly or indirectly)
      refer to the module being deviated.

      Robust clients may want to make sure that they handle a
      situation where a module deviates itself (directly or
      indirectly) gracefully.";
  }
}

grouping module-set-parameters {
  description
    "A set of parameters that describe a module set.";

  leaf index {
    type uint8;
    description
      "An arbitrary number assigned of the module set.";
  }
}
```



```
}
list module {
  key "identifier";
  description
    "An entry in this list represents a module implemented
    by the server, as per RFC 7950 section 5.6.5, with a
    particular set of supported features and deviations.";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";

  uses module-identification-leafs;

  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.
      This leaf is present in the model to keep the alignment
      with 'ietf-yang-library'. Support of this leaf in
      constrained devices is not required, nor expected.";
  }

  uses location-leaf-list;

  list submodule {
    key "identifier";
    description
      "Each entry represents one submodule within the parent
      module.";
    uses module-identification-leafs;
    uses location-leaf-list;
  }

  uses implementation-parameters;
}
list import-only-module {
  key "identifier revision";
  description
    "An entry in this list indicates that the server imports
    reusable definitions from the specified revision of the
    module, but does not implement any protocol accessible
    objects from this revision.

    Multiple entries for the same module name MAY exist.
    This can occur if multiple modules import the same
    module, but specify different revision-dates in the
    import statements.";

  leaf identifier {
```

```
    type sid:sid;
    description
      "The YANG module name.";
  }
  leaf revision {
    type union {
      type revision-identifier;
      type string {
        length 0;
      }
    }
    description
      "The YANG module revision date.";
  }
  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.
      This leaf is present in the model to keep the alignment
      with 'ietf-yang-library'. Support of this leaf in
      constrained devices is not required, nor expected.";
  }

  uses location-leaf-list;

  list submodule {
    key "identifier";
    description
      "Each entry represents one submodule within the
      parent module.";

    uses module-identification-leafs;
    uses location-leaf-list;
  }
}

grouping yang-library-parameters {
  description
    "The YANG library data structure is represented as a grouping
    so it can be reused in configuration or another monitoring
    data structure.";

  list module-set {
    key index;
    description
      "A set of modules that may be used by one or more schemas.
```

```

    A module set does not have to be referentially complete,
    i.e., it may define modules that contain import statements
    for other modules not included in the module set.";

    uses module-set-parameters;
}

list schema {
    key "index";
    description
        "A datastore schema that may be used by one or more
        datastores.

        The schema must be valid and referentially complete,
        i.e., it must contain modules to satisfy all used import
        statements for all modules specified in the schema.";

    leaf index {
        type uint8;
        description
            "An arbitrary reference number assigned to the schema.";
    }
    leaf-list module-set {
        type leafref {
            path "../../module-set/index";
        }
        description
            "A set of module-sets that are included in this schema.
            If a non import-only module appears in multiple module
            sets, then the module revision and the associated
            features and deviations must be identical.";
    }
}

list datastore {
    key "identifier";
    description
        "A datastore supported by this server.

        Each datastore indicates which schema it supports.

        The server MUST instantiate one entry in this list
        per specific datastore it supports.

        Each datastore entry with the same datastore schema
        SHOULD reference the same schema.";

    leaf identifier {
```

```
        type ds:datastore-ref;
        description
            "The identity of the datastore.";
    }
    leaf schema {
        type leafref {
            path "../../schema/index";
        }
        mandatory true;
        description
            "A reference to the schema supported by this datastore.
            All non import-only modules of the schema are
            implemented with their associated features and
            deviations.";
    }
}

/*
 * Top-level container
 */

container yang-library {
    config false;
    description
        "Container holding the entire YANG library of this server.";

    uses yang-library-parameters;

    leaf checksum {
        type binary;
        mandatory true;
        description
            "A server-generated checksum or digest of the contents of
            the 'yang-library' tree. The server MUST change the
            value of this leaf if the information represented by
            the 'yang-library' tree, except 'yang-library/checksum',
            has changed.";
    }
}

/*
 * Notifications
 */

notification yang-library-update {
    description
        "Generated when any YANG library information on the
```

```
        server has changed.";

    leaf checksum {
        type leafref {
            path "/yanglib:yang-library/yanglib:checksum";
        }
        mandatory true;
        description
            "Contains the YANG library checksum or digest for the
             updated YANG library at the time the notification is
             generated.";
    }
}
}
}
<CODE ENDS>
```

5. IANA Considerations

5.1. YANG Module Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950].

name: ietf-constrained-yang-library

namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library

prefix: lib

reference: RFC XXXX

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

6. Security Considerations

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access to these data nodes.

Specifically, the 'module' list may help an attacker to identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker to identify server

implementations with such a defect, in order to launch a denial of service attack on these devices.

7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, `ietf-yang-library` [RFC8525], we will like to thanks to the authors of this YANG module. A special thank also to Andy Bierman for his initial recommendations for the creation of this YANG module.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [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/info/rfc8342>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.

8.2. Informative References

- [I-D.ietf-core-sid] Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.

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

Authors' Addresses

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

Email: michel.veillette@trilliantinc.com

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

Email: ivaylo@ackl.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 23, 2020

A. Keranen
Ericsson
C. Amsuess
July 22, 2019

SenML Base Name Prefix Indication
draft-keranen-core-senml-base-prefix-00

Abstract

The Sensor Measurement Lists (SenML) media type uses globally unique names to facilitate information exchange across systems. This requirement often leads to long names and hence large amount of data to be transmitted with each SenML Pack. This document defines an efficient mechanism to indicate a globally unique prefix for names.

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 January 23, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Indicating base name prefix	3
3.1. IP address	4
3.2. Request Base URI	4
3.3. Public key fingerprint	4
3.4. TLS PSK Identity	5
3.5. CoRE Resource Directory endpoint ID	5
4. IANA Considerations	5
Acknowledgements	6
6. References	6
6.1. Normative References	6
6.2. Informative References	7
Authors' Addresses	7

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] indicates sources of information with globally unique names. A SenML Pack can indicate "base names" ("bn" field) that are prefixed to all names ("n" fields) in the SenML Records where the "bn" appears and all subsequent records until a new base name is indicated. Example of a SenML Pack with two sources of measurement data that share a base name is shown in Figure 1.

```
[
  {"bn":"2001:db8:1234:5678::1/",
   "n":"temperature", "u":"Cel", "v":25.2},
  {"n":"humidity", "u":"%RH", "v":30}
]
```

Figure 1: SenML Pack with two sources of measurements

The base name construct enables indicating the globally unique part of the name only once for a set of Records. However, the globally unique part still needs to be indicated in each Pack. Since base name is often relatively long, in some scenarios it would be useful to be able to further compress or omit this information.

Since the sender and receiver of a SenML Pack often share context information beyond what is in the SenML Pack, e.g., request URI when a RESTful protocol is used, sender IP address, or security association information. This information can be used to assist constructing the globally unique part of a name. However, the sender of the Pack needs to be able to indicate unambiguously what

information is used and how the name is generated from that information.

This document registers a new SenML field to indicate what information outside of the SenML Pack should be used as a prefix to the base name(s). Also rules for consistently creating SenML names from this information is specified for each information source.

Since SenML has only a small set of characters that are allowed in names (see Section 4.5.1 in [RFC8428]) replacing rules for characters outside of this set (e.g., brackets and semicolons) are also defined.

2. Terminology

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.

Readers should also be familiar with the terms and concepts discussed in [RFC8428].

3. Indicating base name prefix

SenML field "bpi" contains an integer value that indicates what information should be used by the receiver to construct a prefix to the base names. This document defines following modes:

- 1: IPv4 or IPv6 address of the sender
- 2: IPv4 or IPv6 address and port of the sender
- 3: Base URI of the request URI
- 5: Fingerprint of the public key of the sender
- 6: TLS PSK Identity
- 10: CoRE Resource Directory endpoint ID

The following sub-sections define rules for generating SenML basename prefix for each mode. Some rules require replacing characters from the input identifiers with characters that are safe for SenML names. Applying these rules may result in multiple different input identifiers being mapped to the same output identifier. The sender of the SenML Pack MUST ensure that such mapping does not result in conflicting names from that sender.

3.1. IP address

The SenML Pack sender IP address can be used without (mode 1) or with (mode 2) port number to generate the base name prefix.

For IPv6 addresses the format defined in [RFC5952] without brackets ("[" and "]") MUST be used.

Example: "2001:db8::1"

For IPv6 address and port underscore ("_") for port separator MUST be used.

Example: "2001:db8::1_5683"

For IPv4 addresses the "dot decimal" notation MUST be used.

Example: "192.0.2.1"

For IPv4 addresses colon (":") MUST be used for port separator.

Example: "192.0.2.1:5684"

3.2. Request Base URI

When a RESTful protocol (e.g., CoAP [RFC7252] or HTTP [RFC7230]) is used to request a SenML Pack, the base of the target request URI can be used as the base name prefix. If IP address is used in the authority part of the URI, rules in Section 3.1 MUST be followed for it.

Characters in the URI that are not in the character set allowed for SenML names MUST be replaced with the underscore ("_") character.

Example: "coap://example.com/room1/"

3.3. Public key fingerprint

When X.509 certificate or raw public key is used to setup the security association (e.g., a TLS connection) to retrieve a SenML Pack, a fingerprint of the public key of the sender of the SenML Pack can be used as the base name prefix. For a public key fingerprint the "URL Segment Format" of [RFC6920] with ";" characters replaced with "_" MUST be used.

Example: "sha-256_UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q"

3.4. TLS PSK Identity

When a pre-shared key (PSK) ciphersuites (e.g., [RFC4279])) are used to establish a TLS connection, the PSK identity can be used as the base name prefix. In this mode the PSK identity from the TLS handshake with characters not allowed for SenML names replaced with "_" MUST be used.

Example: "foo.example.com"

3.5. CoRE Resource Directory endpoint ID

When the sender has registered with the receiving system using the CoRE Resource Directory [I-D.ietf-core-resource-directory] interface, and has defined an "endpoint name" during the registration, the endpoint name can be used as the base name prefix. Characters not allowed for SenML names MUST be replaced with "_".

Example: "urn:dev:ow:10e2073a01080063"

4. IANA Considerations

IANA is requested to assign a new label in the "SenML Labels" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]) for the "Base Name Prefix Indicator" as follows:

Name	Label	JSON Type	XML Type	Reference
Base Name Prefix Indicator	bpi	Number	int	this document

IANA is requested to create a new "SenML Base Name Prefix Indicator modes" subregistry to the SenML registry. Initial contents of the subregistry is shown below:

Name	Value	Reference
IP address	1	this document
IP address & port	2	this document
Base URI	3	this document
Public key	5	this document
TLS PSK ID	6	this document
CoRE RD endpoint	10	this document

Acknowledgements

TBD

6. References

6.1. Normative References

- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.
- [IANA.senml]
IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005,
<<https://www.rfc-editor.org/info/rfc4279>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010,
<<https://www.rfc-editor.org/info/rfc5952>>.

- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

6.2. Informative References

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Christian Amsuess
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

CoRE Working Group
Internet-Draft
Updates: 7252, 7390, 7641 (if approved)
Intended status: Standards Track
Expires: May 7, 2020

M. Tiloca
R. Hoeglund
RISE AB
C. Amsuess

F. Palombini
Ericsson AB
November 04, 2019

Observe Notifications as CoAP Multicast Responses
draft-tiloca-core-observe-multicast-notifications-01

Abstract

The Constrained Application Protocol (CoAP) allows clients to "observe" resources at a server, and receive notifications as unicast responses upon changes of the resource state. In some use cases, such as based on publish-subscribe, it would be convenient for the server to send a single notification to all the clients observing a same target resource. This document defines how a CoAP server sends observe notifications as response messages over multicast, by synchronizing all the observers of a same resource on a same shared Token value. Besides, this document defines how Group OSCORE can be used to protect multicast notifications end-to-end from the CoAP server to the multiple observer clients.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Server-Side Requirements	4
3. Client-Side Requirements	6
4. Example	7
5. Protection of Multicast Notifications with Group OSCORE	9
5.1. OSCORE Group in Informative Response	9
5.2. Server-Side Requirements	11
5.3. Client-Side Requirements	12
6. Example with Group OSCORE	12
7. Security Considerations	14
8. IANA Considerations	15
9. References	15
9.1. Normative References	15
9.2. Informative References	16
Acknowledgments	17
Authors' Addresses	17

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been extended with a number of mechanisms, including resource Observation [RFC7641]. This enables CoAP clients to register at a CoAP server as "observers" of a resource, and hence being automatically notified with an unsolicited response upon changes of the resource state.

CoAP supports group communication over IP multicast [RFC7390], and [I-D.dijk-core-groupcomm-bis] has been enabling Observe registration requests over multicast, in order for clients to efficiently register as observers of a resource hosted at multiple servers.

However, in a number of use cases, using multicast messages for responses would also be desirable. That is, it would be useful that a server sends observe notifications for a same target resource to multiple observers as responses over IP multicast.

For instance, in CoAP publish-subscribe [I-D.ietf-core-coap-pubsub], multiple clients can subscribe to a topic, by observing the related resource hosted at the responsible broker. When a new value is published on that topic, it would be convenient for the broker to send a single multicast notification at once, to all the subscriber clients observing that topic.

A different use case concerns clients observing a same registration resource at the CoRE Resource Directory [I-D.ietf-core-resource-directory]. For example, multiple clients can benefit of observation for discovering (to-be-created) OSCORE groups [I-D.ietf-core-oscore-groupcomm], by retrieving from the Resource Directory updated links and descriptions to join them through the respective Group Manager [I-D.tiloca-core-oscore-discovery].

More in general, multicast notifications would be beneficial whenever several CoAP clients observe a same target resource at a CoAP server, and can be all notified at once by means of a single response message. However, CoAP does not currently define response messages over IP multicast. This specification fills this gap and provides the following twofold contribution.

First, it defines a method to deliver Observe notifications as CoAP responses over IP multicast. In the proposed method, the group of potential observers entrusts the server to manage the Token space for multicast notifications. By doing so, the server provides all the observers of a target resource with the same Token value to bind to their own observation. That Token value is then used in every multicast notification for that target resource. This is achieved by means of an informative unicast response sent by the server to each observer client.

Second, this specification defines how to use Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect multicast notifications end-to-end between the server and the observer clients. This is also achieved by means of the informative unicast response mentioned above, which additionally includes parameter values used by the server to secure every multicast notification for the target resource by using Group OSCORE. This provides a secure binding between each of such notifications and the observation of each of the clients.

1.1. Terminology

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.

Readers are expected to be familiar with terms and concepts described in CoAP [RFC7252], group communication for CoAP [RFC7390] [I-D.dijk-core-groupcomm-bis], Observe [RFC7641], CBOR [RFC7049], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This specification additionally defines the following terminology.

- o Traditional observation. A resource observation associated to a single observer client, as defined in [RFC7641].
- o Group observation. A resource observation associated to a group of clients. The server sends notifications for the group-observed resource over IP multicast to all the observer clients.
- o Phantom request. The CoAP request message that the server would have received to generate a group observation on one of its resources. The phantom request is generated inside the server and does not hit the wire.
- o Informative response. A CoAP response message that the server sends to a given client via unicast, providing the client with information on a group observation.

2. Server-Side Requirements

The server can, at any time, start a group observation on one of its resources. Practically, the server may want to do that under the following circumstances.

- o In the absence of observations for the target resource, the server receives a registration request from a first client wishing to start a traditional observation on that resource.
- o When a certain amount of traditional observations has been established on the target resource, the server decides to make those clients part of a group observation on that resource.

When it wants to start a group observation on one of its resources, and assuming it knows the multicast IP address to use to send multicast notifications to, the server proceeds as follows.

1. The server builds a phantom observation request, i.e. a GET request with an Observe option set to 0 (register).
2. The server selects a currently available value T, from the token space used for messages from the chosen multicast IP address to the server address intended for accessing the target resource. That token space is under exclusive control of the server.
3. The server processes the phantom observation request above, without transmitting it on the wire. The request is addressed to the resource for which the server wants to start the group observation, as if sent from the group of observers, i.e. with the multicast IP address as source address.
4. Upon processing the self-generated phantom request, the server interprets it as an observe registration received from the group of potential observer clients. In particular, from then on, the server MUST use T as its own local Token value associated to that observation, with respect to the (next hop towards the) clients.
5. The server does not immediately respond to the phantom observation request with a multicast notification. The server stores the phantom observation request as is, throughout the lifetime of the group observation.

After having started a group observation on a target resource, the server proceeds as follows.

- o For each traditional observation ongoing on the target resource, the server MAY cancel that observation, and then adds the corresponding client to the list of observers taking part in the group observation. The server sends to each of such clients an informative response message, encoded as a unicast response with response code 5.03 (Service Unavailable). As per [RFC7641], such a response does not include an Observe option. The response MUST be Confirmable and MUST NOT encode link-local addresses. The payload of the response is a CBOR map including the following fields.
 - * The IP multicast address where the server will hereafter send multicast notifications for the target resource, encoded as a CBOR byte string, with CBOR label "address".

- * The byte serialization of the phantom observation request received by the server, encoded as a CBOR byte string, with CBOR label "registr".
 - * Optionally, the current representation of the target resource, encoded as a CBOR byte string, with CBOR label "res".
 - o Upon receiving a registration request to observe the target resource, the server does not create a corresponding individual observation for the requesting client. Instead, the server adds that client to the list of observers taking part in the group observation of the target resource, and replies to the client with the same informative response message defined above, which **MUST** be Confirmable and **MUST** include the current representation of the target resource. Note that this also applies when, with no ongoing traditional observations on the target resource, the server receives a registration request from a first client and decides to start a group observation on the target resource.
 - o Upon a change of the status of the target resource, the server sends a multicast notification, intended to all the clients taking part in the group observation of that resource. In particular, each of such multicast notifications:
 - * **MUST** be sent to the IP multicast address indicated in the informative response message to the observer clients.
 - * **MUST** include an Observe option, as per [RFC7641].
 - * **MUST** have the same Token value T of the phantom registration request that started the group observation, also included in the informative response message to the observer clients. That is, every multicast notification for a target resource is not bound to the observation requests from the different clients, but rather to the phantom registration associated to the whole set of clients currently taking part in the group observation of that resource.
3. Client-Side Requirements

A client sends an observation request to the server as described in [RFC7641], i.e. a GET request with an Observe option set to 0 (register). The request **MUST NOT** encode link-local addresses. If the server is not configured to accept registrations on that target resource with a group observation, this would still result in a positive notification response to the client as described in [RFC7641].

Upon receiving the informative response defined in Section 2, the client proceeds as follows.

1. The client configures an observation of the target resource from a CoAP endpoint associated to the multicast IP address, and stores the phantom registration request specified in the informative response from the server. The group observation is bound to the phantom registration request. In particular, the client **MUST** use its Token value T as its own local Token value associated to that group observation, with respect to the (next hop towards the) server. The particular way to achieve this is implementation specific.
2. If a traditional observation to the target resource is ongoing, the client **MAY** silently cancel it without notifying the server.
3. If the informative response from the server includes the current representation of the target resource, the client considers it as received from an observe notification and processes it as usual.

From then on, the client will receive, accept and process multicast notifications about the state of the target resource from the server, as responses to the phantom registration request and with Token value T.

4. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S. Before the following exchanges occur, no clients are observing the resource /r , which has value "1234".

The server S sends multicast notifications to the IP multicast address M_addr , and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

```

C_1      ----- [ Unicast ] -----> S    /r
|
| GET
| Token: 0x4a
| Observe: 0 (Register)
|
|                                     (S allocates the available Token value 0xff .)
|

```

```

(S sends to itself a phantom observation request Ph_req
as coming from the IP multicast address M_addr .)
-----
/
\-----> /r
                GET
                Token: 0xff
                Observe: 0 (Register)

(S creates a group observation of /r .)

(S adds C_1 to the list of observers taking
part in the group observation of /r .)

C_1 <----- [ Unicast ] ----- S
5.03
Token: 0x4a
Payload: { address : M_addr ,
          registr : Ph_req ,
          res : "1234" }

C_2 ----- [ Unicast ] -----> S /r
GET
Token: 0x01
Observe: 0 (Register)

(S adds C_2 to the list of observers taking
part in the group observation of /r .)

C_2 <----- [ Unicast ] ----- S
5.03
Token: 0x01
Payload: { address : M_addr ,
          registr : Ph_req ,
          res : "1234" }

(The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast ] ----- S
C_2 (Destination address: M_addr)
2.05
Token: 0xff
Observe: 11
Payload: "5678"

```

5. Protection of Multicast Notifications with Group OSCORE

A server can protect multicast notifications by using Group OSCORE [I-D.ietf-core-oscore-groupcomm]. In such a case, both the server and the clients interested in receiving multicast notifications from that server have to be members of the same OSCORE group.

Clients MAY discover the OSCORE group to refer to by using the method in [I-D.tiloca-core-oscore-discovery], also based on the CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory]. Alternatively, the server MAY communicate to the client what OSCORE group to join, as described in Section 5.1. Furthermore, both the clients and the server MAY join the OSCORE group by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz]. Further details on how to discover the OSCORE group and join it are out of the scope of this specification.

Alternative security protocols than Group OSCORE, such as OSCORE [RFC8613] and/or DTLS [RFC6347][I-D.ietf-tls-dtls13], can be used to protect other exchanges via unicast between the server and each client, including the original client registration (see Section 3).

5.1. OSCORE Group in Informative Response

This section describes a mechanism for the server to communicate to the client what OSCORE group to join in order to decrypt and verify the multicast notifications protected with group OSCORE. The client MAY use the information provided by the server to start the ACE joining procedure described in [I-D.ietf-ace-key-groupcomm-oscore]. This mechanism is OPTIONALLY supported by client and server.

Additionally to what defined in Section 2, the CBOR map in the informative response payload contains the following fields:

- o The URI for joining the OSCORE group at the respective Group Manager, encoded as a CBOR text string, with CBOR label "join-uri". If the procedure described in [I-D.ietf-ace-key-groupcomm-oscore] is used for joining, this field specifically indicates the URI of the group-membership resource at the Group Manager.
- o The name of the OSCORE group, encoded as a CBOR text string, with CBOR label "sec-gp".

- o Optionally, the URI of the Authorization Server associated to the Group Manager for the OSCORE group, encoded as a CBOR text string, with CBOR label "as-uri".
- o Optionally, the algorithm used to countersign messages, encoded as a text string or an int, with value taken from the 'Value' column of the "COSE Algorithms" registry defined in [RFC8152], with CBOR label "cs-alg".
- o Optionally, the elliptic curve for the algorithm used to countersign messages, encoded as a text string or an int, with value taken from the 'Value' column of the "COSE Elliptic Curve" registry defined in [RFC8152], with CBOR label "cs-crv".
- o Optionally, the key type of countersignature keys used to countersign messages, encoded as a text string or an int, with value taken from the 'Value' column of the "COSE Key Types" registry defined in [RFC8152], with CBOR label "cs-kty".
- o Optionally, the encoding of the public keys, encoded as an int, with value taken from the 'Confirmation Key' column of the "CWT Confirmation Method" registry defined in [I-D.ietf-ace-cwt-proof-of-possession], with CBOR label "cs-kenc". Future specifications may define additional values for this parameter.
- o Optionally, the AEAD algorithm, encoded as a text string or an int, with value taken from the 'Value' column of the "COSE Algorithms" registry defined in [RFC8152], with CBOR label "alg".
- o Optionally, the HKDF algorithm, encoded as a text string or an int, with value taken from the 'Value' column of the "COSE Algorithms" registry defined in [RFC8152], with CBOR label "hkdf".

The values of 'cs_alg', 'cs_crv', 'cs_kty' and 'cs_kenc' provide an early knowledge of the format and encoding of public keys used in the OSCORE group. Thus, the client does not need to ask the Group Manager for this information as a preliminary step before the (ACE) join process, or to perform a trial-and-error exchange with the Group Manager upon joining the group. Hence, the client is able to provide the Group Manager with its own public key in the correct expected format and encoding, at the very first step of the (ACE) join process.

The values of 'cs_alg', 'alg' and 'hkdf' provide an early knowledge of the algorithms used in the OSCORE group. Thus, the client is able to decide whether to actually proceed with the (ACE) join process, depending on its support for the indicated algorithms.

As mentioned above, since this mechanism is OPTIONAL, all the fields are OPTIONAL in the informative response. However, the "join-uri" and "sec-gp" fields MUST be present if the mechanism is implemented and run. If any of the fields are present without the "join-uri" and "sec-gp" fields present, the client MUST ignore these fields, since they would not be sufficient to start the (ACE) join procedure. When this happens, the client MAY try sending a new registration request to the server (see Section 3). Otherwise, the client SHOULD explicitly withdraw from the group observation, as defined in Section 3.

5.2. Server-Side Requirements

When using Group OSCORE to protect multicast notifications, the server performs as described in Section 2, with the following differences.

- o The phantom registration request MUST be secured, by using Group OSCORE. To this end, the server protects the phantom registration request as if it was the actual sender, i.e. by using its own Sender Context. As a consequence, the server consumes the current value of its own Sender Sequence Number SN in the OSCORE group, and hence updates it to $SN^* = (SN + 1)$. Consistently, the OSCORE option in the phantom registration request includes:
 - * As 'kid', the Sender ID of the server in the OSCORE group.
 - * As 'piv', the previously consumed sender sequence number value SN of the server in the OSCORE group, i.e. $(SN^* - 1)$.
- o Upon sending every multicast notification for the target resource, the server protects it with Group OSCORE. In particular, the process described in Section 6.3 of [I-D.ietf-core-oscore-groupcomm] applies, with the following additions when building the two OSCORE 'external_aad' to encrypt and countersign the multicast notification (see Sections 3.1 and 3.2 of [I-D.ietf-core-oscore-groupcomm]).
 - * The 'request_kid' is the 'kid' value in the OSCORE option of the phantom registration request, i.e. the Sender ID of the server.
 - * The 'request_piv' is the 'piv' value in the OSCORE option of the phantom registration request, i.e. the consumed sender sequence number SN of the server.

5.3. Client-Side Requirements

When using Group OSCORE to protect multicast notifications, the client performs as described in Section 3, with the following differences.

- o Upon receiving the informative response from the server, the client retrieves the specified phantom registration request, and extracts the 'kid' and 'piv' fields of its OSCORE option.
- o From then on, when verifying multicast notifications for that target resource as described in Section 6.4 of [I-D.ietf-core-oscore-groupcomm], the client MUST use the extracted 'kid' as 'request_kid' and the extracted 'piv' as 'request_piv', in the two 'external_aad' for decrypting and verifying every multicast notification from the server for the target resource (see Sections 3.1 and 3.2 of [I-D.ietf-core-oscore-groupcomm]). The particular way to achieve this is implementation specific.

6. Example with Group OSCORE

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S. Before the following exchanges occur, no clients are observing the resource /r, which has value "1234".

The server S sends multicast notifications to the IP multicast address M_addr, and starts the group observation upon receiving a registration request from a first client that wishes to start a traditional observation on the resource /r.

Pairwise communication over unicast are protected with OSCORE, while S protects multicast notifications with Group OSCORE. Specifically:

- o C_1 and S have a pairwise OSCORE Security Context. In particular, C_1 has 'kid' = 1 as Sender ID, and SN_1 = 101 as Sequence Number. Also, S has 'kid' = 3 as Sender ID, and SN_3 = 301 as Sequence Number.
- o C_2 and S have a pairwise OSCORE Security Context. In particular, C_2 has 'kid' = 2 as Sender ID, and SN_2 = 201 as Sequence Number. Also, S has 'kid' = 4 as Sender ID, and SN_4 = 401 as Sequence Number.
- o S is a member of the OSCORE group with name "myGroup", and 'kid_context' = "feedca57ab2e" as Group ID. In the OSCORE group, S has 'kid' = 5 as Sender ID, and SN_5 = 501 as Sequence Number.

```

C_1      ----- [ Unicast w/ OSCORE ] -----> S    /r
GET
Token: 0x4a
Observe: 0 (Register)
OSCORE: {kid: 1 ; piv: 101 ; ...}

      (S allocates the available Token value 0xff .)

      (S sends to itself a phantom observation request Ph_req
      as coming from the IP multicast address M_addr .)
      -----
      /
      \-----> /r

                        GET
                        Token: 0xff
                        Observe: 0 (Register)
                        OSCORE: {kid: 5 ; piv: 501 ; ...}

      (S steps SN_5 in the Group OSCORE Sec. Ctx : SN_5 <= 502)

                        (S creates a group observation of /r .)

                        (S adds C_1 to the list of observers taking
                        part in the group observation of /r .)

C_1 <----- [ Unicast w/ OSCORE ] ----- S
5.03
Token: 0x4a
OSCORE: {piv: 301; ...}
Payload: { address  : M_addr ,
          registr  : Ph_req ,
            res    : "1234" ,
          join-uri : coap://myGM/group-oscore/myGroup
          sec-gp   : "myGroup" }

C_2      ----- [ Unicast w/ OSCORE ] -----> S    /r
GET
Token: 0x01
Observe: 0 (Register)
OSCORE: {kid: 2 ; piv: 201 ; ...}

      (S adds C_2 to the list of observers taking
      part in the group observation of /r .)

C_2 <----- [ Unicast w/ OSCORE ] ----- S

```

```

5.03
Token: 0x01
OSCORE: {piv: 401; ...}
Payload: { address  : M_addr ,
           registr  : Ph_req ,
           res      : "1234" ,
           join-uri : coap://myGM/group-oscore/myGroup
           sec-gp   : "myGroup" }

           (The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast w/ Group OSCORE ] ----- S
C_2           (Destination address: M_addr)
2.05
Token: 0xff
Observe: 11
OSCORE: {kid: 5; piv: 502 ; ...}
Payload: "5678"

```

The two external_aad used to encrypt and countersign the multicast notification above have 'req_kid' = 5 and 'req_iv' = 501. These are indicated in the 'kid' and 'iv' field of the OSCORE option of the phantom observation request, which is included in the 'registr' parameter of the unicast informative response to the two clients. Thus, the two clients can build the two same external_aad for decrypting and verifying this multicast notification and the following ones.

7. Security Considerations

The same security considerations from [RFC7252][RFC7390][RFC7641][I-D.dijk-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast (notification) responses MUST also be secured, including and especially the informative responses from the server. This prevents on-path active adversaries from altering the conveyed IP multicast address and serialized phantom request. Thus, it ensures secure binding between every multicast notification for a same observed resource and the phantom request that started the group observation of that resource.

To this end, clients and servers SHOULD use OSCORE or Group OSCORE, so ensuring that the secure binding above is enforced end-to-end between the server and each observing client.

8. IANA Considerations

This document has the following actions for IANA.

TODO: registry for labels of the map in the informative response.

9. References

9.1. Normative References

- [I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-01 (work in progress), July 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-06 (work in progress), November 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [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/info/rfc8613>>.

9.2. Informative References

- [I-D.ietf-ace-cwt-proof-of-possession]
Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-11 (work in progress), October 2019.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-03 (work in progress), November 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-25 (work in progress), October 2019.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-09 (work in progress), September 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-33 (work in progress), October 2019.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-03 (work in progress), July 2019.

- [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/info/rfc6347>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, John Mattsson, Ludwig Seitz, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

M. Tiloca
RISE AB
C. Amsuess

P. van der Stok
Consultant
November 04, 2019

Discovery of OSCORE Groups with the CoRE Resource Directory
draft-tiloca-core-oscore-discovery-04

Abstract

Group communication over the Constrained Application Protocol (CoAP) can be secured by means of Group Object Security for Constrained RESTful Environments (Group OSCORE). At deployment time, devices may not know the exact OSCORE groups to join, the respective Group Manager, or other information required to perform the joining process. This document describes how a CoAP endpoint can use descriptions and links of resources registered at the CoRE Resource Directory to discover OSCORE groups and to acquire information for joining them through the respective Group Manager. A given OSCORE group may protect multiple application groups, which are separately announced in the Resource Directory as sets of endpoints sharing a pool of resources. This approach is consistent with, but not limited to, the joining of OSCORE groups based on the ACE framework for Authentication and Authorization in constrained environments.

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 May 7, 2020.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Registration Resource for Group Managers	5
3. Registration of Group Manager Endpoints	6
4. Addition and Update of OSCORE Groups	8
5. Discovery of OSCORE Groups	9
5.1. Discovery Example #1	10
5.2. Discovery Example #2	11
6. Use Case Example With Full Discovery	13
7. Security Considerations	17
8. IANA Considerations	17
8.1. Resource Types	17
9. References	18
9.1. Normative References	18
9.2. Informative References	19
Acknowledgments	20
Authors' Addresses	20

1. Introduction

A set of CoAP endpoints constitutes an application group by sharing a common pool of resources. The members of an application group may be members of a given security group, by sharing a common set of keying material to secure group communication.

The Constrained Application Protocol (CoAP) [RFC7252] supports group communication over IP multicast [RFC7390][I-D.dijk-core-groupcomm-bis] to improve efficiency and latency of communication and reduce bandwidth requirements. The document Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes how to achieve end-to-end security for

CoAP messages through CBOR Object Signing and Encryption (COSE) [RFC8152].

In particular, [I-D.ietf-core-oscore-groupcomm] specifies how Group OSCORE protects CoAP messages in group communication contexts, so enabling OSCORE groups as security groups. Typically, one application group relies on exactly one OSCORE group, while a same OSCORE group may be used by multiple application groups at the same time.

A CoAP endpoint joins an OSCORE group via a Group Manager (GM), in order to get the necessary group keying material. As in [I-D.ietf-ace-key-groupcomm-oscore], the joining process can be based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], with the joining endpoint and the GM acting as ACE Client and Resource Server, respectively. That is, the joining endpoint accesses the group-membership resource associated with the OSCORE group to join and exported by the GM.

Typically, devices are equipped with a static X509 IDevID certificate installed at manufacturing time. This certificate is used at deployment time during an enrollment process that provides the device with an Operational Certificate, possibly updated during the device lifetime. In the presence of secure group communication for CoAP, such an Operational Certificate may be accompanied by information required to join OSCORE groups. This especially includes a reference to the group-membership resources to access at the respective GMs.

However, it is usually impossible to provide such precise information to freshly deployed devices as part of their (early) Operational Certificate. This can be due to a number of reasons: (1) the OSCORE group(s) to join and the responsible GM(s) are generally unknown at manufacturing time; (2) an OSCORE group of interest is created, or the responsible GM is deployed, only after the device is enrolled and fully operative in the network; and (3) information related to existing OSCORE groups or to their GMs has been changed. This requires a method for CoAP endpoints to dynamically discover OSCORE groups and their GM, and to retrieve relevant information about deployed groups.

To this end, CoAP endpoints can use descriptions and links of group-membership resources at GMs, in order to discover OSCORE groups and to retrieve the information required for joining them. With the discovery process of OSCORE groups expressed in terms of links to resources, the remaining problem is thus the discovery of those links. The CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory] provides such discovery in an

efficient way, and it is expected to be used in many setups that would benefit of OSCORE group discovery.

This specification builds on this approach and describes how CoAP endpoints can use the RD to carry out the necessary link discovery steps, in order to discover OSCORE groups of interest and retrieve the information required to join them through their GM. In principle, the GM registers as an endpoint with the RD. The corresponding registration resource includes one link for each OSCORE group under that GM, specifying the path to the related group-membership resource to access for joining that group.

More information about the OSCORE group is stored in the target attributes of the respective link. This especially includes the identifiers of the application groups which use that OSCORE group. This enables a lookup of those application groups at the Resource Directory, where they are separately announced by a Commissioning Tool (see Appendix A of [I-D.ietf-core-resource-directory]).

When querying the RD for OSCORE groups, a CoAP endpoint can further benefit of the CoAP Observe Option [RFC7641]. This enables the reception of notifications about the creation of new OSCORE groups or the updates concerning existing groups. Thus, it facilitates the early deployment of CoAP endpoints, i.e. even before the GM is deployed and OSCORE groups are created.

The approach in this document is consistent with, but not limited to, the joining of OSCORE groups in [I-D.ietf-ace-key-groupcomm-oscore].

1.1. Terminology

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.

This specification requires readers to be familiar with the terms and concepts discussed in [I-D.ietf-core-resource-directory] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252], [I-D.ietf-core-oscore-groupcomm] and [I-D.ietf-ace-key-groupcomm-oscore].

Terminology for constrained environments, such as "constrained device" and "constrained-node network", is defined in [RFC7228].

This document also refers to the following terminology.

- o OSCORE group: a set of CoAP endpoints that share one OSCORE Common Security Context to protect group communication as described in [I-D.ietf-core-oscore-groupcomm]. Consequently, an OSCORE group acts as security group for all its members.
- o Application group: a set of CoAP endpoints that share a set of common resources. Application groups are announced in the RD by a Commissioning Tool, according to the RD-Groups usage pattern (see Appendix A of [I-D.ietf-core-resource-directory]). An application group can be associated with a single OSCORE group, while multiple application groups can use the same OSCORE group. Application groups share resources by definition. Any two application groups associated to the same OSCORE group do not share any same resource.

2. Registration Resource for Group Managers

With reference to Figure 3 of [I-D.ietf-core-resource-directory], a Group Manager (GM) registers as an endpoint with the CoRE Resource Directory (RD). The registration includes the links to the group-membership resources located at the GM, and associated to the OSCORE groups administrated by that GM.

In particular, each link to a group-membership resource includes:

- o "target": URI of the group-membership resource at the GM.
- o target attributes, including:
 - * Resource Type (rt) with the value "core.osc.mbr" defined in Section 8.1 of this specification.
 - * The name of the OSCORE group, as defined in [I-D.ietf-ace-key-groupcomm-oscore].
 - * One target attribute for each application group associated with the OSCORE group, specifying the name of that application group.
 - * The algorithm used to countersign messages in the OSCORE group.
 - * The elliptic curve (if applicable) for the algorithm used to countersign messages in the OSCORE group.
 - * The key type of countersignature keys used to countersign messages in the OSCORE group.
 - * The encoding of public keys used in the OSCORE group.

- * The AEAD algorithm used in the OSCORE group.
- * The HKDF algorithm used in the OSCORE group.

3. Registration of Group Manager Endpoints

During deployment, a GM finds the RD as described in Section 4 of [I-D.ietf-core-resource-directory]. Afterwards, the GM registers as an endpoint with the RD, as described in Section 5 of [I-D.ietf-core-resource-directory].

When doing so, the GM also registers all the group-membership resources it has at that point in time, i.e. one for each of its OSCORE groups.

For each registered group-membership resource, the GM includes the following parameters in the payload of the registration request.

- o 'rt' = "core.osc.mbr" (see Section 8.1).
- o 'sec-gp', specifying the name of the OSCORE group of interest. This parameter MUST specify a single value.
- o 'app-gp', specifying the name(s) of the application group(s) associated to the OSCORE group of interest indicated by 'sec-gp'. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group. A same application group MUST NOT be specified multiple times.

Also, for each registered group-membership resource, the GM may additionally include the following parameters in the payload of the registration request.

- o 'cs_alg', specifying the algorithm used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry defined in [RFC8152].
- o 'cs_crv', specifying the elliptic curve (if applicable) for the algorithm used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Elliptic Curve" Registry defined in [RFC8152].
- o 'cs_kty', specifying the key type of countersignature keys used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string,

which is taken from the 'Value' column of the "COSE Key Types" Registry defined in [RFC8152].

- o 'cs_kenc', specifying the encoding of the public keys used in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string. This specification explicitly admits the signaling of COSE Keys [RFC8152] as encoding for public keys, which is indicated with "1", as taken from the 'Confirmation Key' column of the "CWT Confirmation Method" Registry defined in [I-D.ietf-ace-cwt-proof-of-possession]. Future specifications may define additional values for this parameter.
- o 'alg', specifying the AEAD algorithm used in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry defined in [RFC8152].
- o 'hkdf', specifying the HKDF algorithm used in the OSCORE group. If present, this parameter MUST specify a single value encoded as a text string, which is taken from the 'Value' column of the "COSE Algorithms" Registry defined in [RFC8152].

Values registered as a string that looks like an integer are not supported by this approach. Therefore, they MUST NOT be advertised through the corresponding parameters above.

A CoAP endpoint that queries the RD to discover OSCORE groups and their group-membership resource to access (see Section 5) would benefit from the information above as follows.

- o The values of 'cs_alg', 'cs_crv', 'cs_kty' and 'cs_kenc' related to a group-membership resource provide an early knowledge of the format and encoding of public keys used in the OSCORE group. Thus, the CoAP endpoint does not need to ask the GM for this information as a preliminary step before the joining process, or to perform a trial-and-error exchange with the GM. Hence, the CoAP endpoint is able to provide the GM with its own public key in the correct expected format and encoding at the very first step of the joining process.
- o The values of 'cs_alg', 'alg' and 'hkdf' related to a group-membership resource provide an early knowledge of the algorithms used in the OSCORE group. Thus, the CoAP endpoint is able to decide whether to actually proceed with the joining process, depending on its support for the indicated algorithms.

The GM SHOULD NOT use the Simple Registration approach described in Section 5.1 of [I-D.ietf-core-resource-directory].

The example below shows a GM with endpoint name "gm1" and address 2001:db8::ab that registers with the RD. The GM specifies the value of the 'sec-gp' parameter for accessing the OSCORE group with name "feedca570000" and used by the application group with name "group1" specified with the value of the 'app-gp' parameter. The countersignature algorithm used in the OSCORE group is EdDSA, with elliptic curve Ed25519 and keys of type OKP. Public keys used in the OSCORE group are encoded as COSE Keys [RFC8152].

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gm1

Content-Format: 40

Payload:

```
</group-oscore/feedca570000>;ct=41;rt="core.osc.mbr";  
    sec-gp="feedca570000";app-gp="group1";  
    cs_alg="-8";cs_crv="6";cs_kty="1";  
    cs_kenc="1"
```

Response: RD -> GM

Res: 2.01 Created

Location-Path: /rd/4521

4. Addition and Update of OSCORE Groups

The GM is responsible to refresh the registration of all its group-membership resources in the RD. This means that the GM has to update the registration within its lifetime as per Section 5.3.1 of [I-D.ietf-core-resource-directory], and has to change the content of the registration when a group-membership resource is added/removed or if its target attributes have to be changed, such as in the following cases.

- o The GM creates a new OSCORE group and starts exporting the related group-membership resource.
- o The GM dismisses an OSCORE group and stops exporting the related group-membership resource.
- o Information related to an existing OSCORE group changes, e.g. the list of associated application groups.

To perform an update of its registrations, the GM can re-register with the RD and fully specify all links to its group-membership resources with their target attributes.

The example below shows how the GM from Section 3 re-registers with the RD. When doing so, it specifies:

- o The same previous group-membership resource associated to the OSCORE group with name "feedca570000".
- o An additional group-membership resource associated to the OSCORE group with name "ech0ech00000" and used by the application group "group2".
- o A third group-membership resource associated with the OSCORE group with name "abcdef120000" and used by two application groups, namely "group3" and "group4".

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: 40

Payload:

```
</group-oscore/feedca570000>;ct=41;rt="core.osc.mbr";
    sec-gp="feedca570000";app-gp="group1";
    cs_alg="-8";cs_crv="6";cs_kty="1";
    cs_kenc="1",
</group-oscore/ech0ech00000>;ct=41;rt="core.osc.mbr";
    sec-gp="ech0ech00000";app-gp="group2";
    cs_alg="-8";cs_crv="6";cs_kty="1";
    cs_kenc="1",
</group-oscore/abcdef120000>;ct=41;rt="core.osc.mbr";
    sec-gp="abcdef120000";app-gp="group3";
    app-gp="group4";cs_alg="-8";
    cs_crv="6";cs_kty="1";cs_kenc="1"
```

Response: RD -> GM

Res: 2.04 Changed

Location-Path: /rd/4521

Alternatively, the GM can perform a PATCH/iPATCH [RFC8132] request to the RD, as per Section 5.3.3 of [I-D.ietf-core-resource-directory]. This requires new media-types to be defined in future standards, to apply a link-format document as a patch to an existing stored document.

5. Discovery of OSCORE Groups

A CoAP endpoint that wants to join an OSCORE group, hereafter called the joining node, might not have all the necessary information at

deployment time. Also, it might want to know about possible new OSCORE groups created afterwards by the respective Group Managers.

To this end, the joining node can perform a resource lookup at the RD as per Section 6.1 of [I-D.ietf-core-resource-directory], to retrieve the missing pieces of information needed to join the OSCORE group(s) of interest. The joining node can find the RD as described in Section 4 of [I-D.ietf-core-resource-directory].

The joining node uses the following parameter value for the lookup filtering.

- o 'rt' = "core.osc.mbr" (see Section 8.1).

The joining node may additionally consider the following parameters for the lookup filtering, depending on the information it has already available.

- o 'sec-gp', specifying the name of the OSCORE group of interest. This parameter MUST specify a single value.
- o 'ep', specifying the registered endpoint of the GM.
- o 'app-gp', specifying the name(s) of the application group(s) associated with the OSCORE group of interest. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group. A same application group MUST NOT be specified multiple times.

Note that, with RD-based discovery, including the 'app-gp' parameter multiple times would result in finding only the group-membership resource that serves all the specified application groups, i.e. not any group-membership resource that serves either. Therefore, a joining node needs to perform N separate queries with different values for 'app-gp', in order to safely discover the (different) group-membership resource(s) serving the N application groups.

5.1. Discovery Example #1

Consistently with the examples in Section 3 and Section 4, the example below considers a joining node that wants to join the OSCORE group associated with the application group "group1", but that does not know the name of the OSCORE group, the responsible GM and the group-membership resource to access.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.mbr&app-gp=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/group-oscore/feedca570000>;rt="core.osc.mbr";
  sec-gp="feedca570000";app-gp="group1";
  cs_alg="-8";cs_crv="6";cs_kty="1";
  cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

To retrieve the multicast IP address used in "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered as per Appendix A of [I-D.ietf-core-resource-directory], with ff35:30:2001:db8::23 as associated multicast IP address.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
?et=core.rd-group&ep=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/501>;ep="group1";et="core.rd-group";
  base="coap://[ff35:30:2001:db8::23]"
```

5.2. Discovery Example #2

Consistently with the examples in Section 3 and Section 4, the example below considers a joining node that wants to join the OSCORE group with name "feedca570000", but that does not know the responsible GM, the group-membership resource to access, and the associated application groups.

The example also shows how the joining node uses CoAP observation [RFC7641], in order to be notified of possible changes to the target attributes of the group-membership resource. This is also useful to handle the case where the OSCORE group of interest has not been created yet, so that the joining node can receive the requested information when it becomes available.

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res
    ?rt=core.osc.mbr&sec-gp=feedca570000
Observe: 0
```

Response: RD -> Joining node

```
Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8::ab]/group-oscore/feedca570000>;rt="core.osc.mbr";
    sec-gp="feedca570000";app-gp="group1";
    cs_alg="-8";cs_crv="6";cs_kty="1";
    cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

Depending on the search criteria, the joining node performing the resource lookup can get large responses. This can happen, for instance, when the lookup request targets all the group-membership resources at a specified GM, or all the group-membership resources of all the registered GMs, as in the example below.

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res?rt=core.osc.mbr
```

Response: RD -> Joining node

```
Res: 2.05 Content
Payload:
<coap://[2001:db8::ab]/group-oscore/feedca570000>;rt="core.osc.mbr";
    sec-gp="feedca570000";app-gp="group1";
    cs_alg="-8";cs_crv="6";cs_kty="1";
    cs_kenc="1";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/group-oscore/ech0ech000000>;rt="core.osc.mbr";
    sec-gp="ech0ech000000";app-gp="group2";
    cs_alg="-8";cs_crv="6";cs_kty="1";
    cs_kenc="1";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/group-oscore/abcdef120000>;rt="core.osc.mbr";
    sec-gp="abcdef120000";app-gp="group3";
    app-gp="group4";cs_alg="-8";cs_crv="6";
    cs_kty="1";cs_kenc="1";anchor="coap://[2001:db8::ab]"
```

Therefore, it is RECOMMENDED that a joining node which performs a resource lookup with the CoAP Observe option specifies the value of the parameter 'sec-gp' in its GET request sent to the RD.

6. Use Case Example With Full Discovery

In this section, the discovery of security groups is described to support the installation process of a lighting installation in an office building. The described process is a simplified version of one of many processes.

Assume the existence of four luminaires that are members of two application groups. In the first application group, the four luminaires receive presence messages and light intensity messages from sensors or their proxy. In the second application group, the four luminaires and several other pieces of equipment receive building state schedules.

Each of the two application groups is associated to a different security group and uses its own dedicated multicast IP address.

The Fairhair Alliance describes how a new device is accepted and commissioned in the network [Fairhair], by means of its certificate stored during the manufacturing process. When commissioning the new device in the installation network, the new device gets a new identity defined by a newly allocated certificate, following the BRSKI specification.

Section 7.3 of [I-D.ietf-core-resource-directory] describes how the Commissioning Tool (CT) assigns an endpoint name based on the CN field, (CN=ACME) and the serial number of the certificate (serial number = 123x, with $3 < x < 8$). Corresponding ep-names ACME-1234, ACME-1235, ACME-1236 and ACME-1237 are also assumed.

It is common practice that locations in the building are specified according to a coordinate system. After the acceptance of the luminaires into the installation network, the coordinate of each device is communicated to the CT. This can be done manually or automatically.

The mapping between location and ep-name is calculated by the CT. For instance, on the basis of grouping criteria, the CT assigns: i) application group "grp_R2-4-015" to the four luminaires; and ii) application group "grp_schedule" to all schedule requiring devices. Also, the device with ep name ACME-123x has been assigned IP address: [2001:db8:4::x]. The RD is assigned IP address: [2001:db8:4:ff]. The used multicast addresses are: [ff05::5:1] and [ff05::5:2].

*** **

The CT defines the application group "grp_R2-4-015", with resource /light and base address [ff05::5:1], as follows.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::5:1]
Payload:
</light>;rt="oic.d.light"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/501
```

Also, the CT defines a second application group "grp_schedule", with resource /schedule and base address [ff05::5:2], as follows.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_schedule&et=core.rd-group&base=coap://[ff05::5:2]
Payload:
</schedule>;rt="oic.r.time.period"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/502
```

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

Consecutively, the CT registers the four devices in the RD (IP address: 2001:db8:4::ff), with their endpoint names and application groups.

For the application group "grp_R2-4-015", four endpoints are specified as follows, with x = 4, 5, 6, 7.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ACME-123x&base=coap://[2001:db8:4::x]&app-gp=grp_R2-4-015
Payload:
</light>;rt="oic.d.light"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/452x
```

For the application group "grp_schedule", four other endpoints are specified as follows, with x = 4, 5, 6, 7.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
 ?ep=ACME-123x&base=coap://[2001:db8:4::x]&app-gp=grp_schedule
Payload:
</schedule>;rt="oic.r.time.period"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/456x

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd?ep=gml&base=coap://[2001:db8::ab]
Payload:
</group-oscore/feedca570000>;ct=41;rt="core.osc.mbr";
 sec-gp="feedca570000";app-gp="grp_R2-4-015",
</group-oscore/feedsc590000>;ct=41;rt="core.osc.mbr";
 sec-gp="feedsc590000";app-gp="grp_schedule"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/4521

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

The device with IP address [2001:db8:4::x] can consequently learn the groups to which it belongs. In particular, it first does an ep lookup to the RD to learn the application groups to which it belongs.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
 ?base=coap://[2001:db8:4::x]

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<rd/452x>;base=coap://[2001:db8:4::x]&ep=ACME-123x&\
    app-gp=grp_R2-4-015,
<rd/456x>;base=coap://[2001:db8:4::x]&ep=ACME-123x&\
    app-gp=grp_schedule
```

To retrieve the multicast IP address used in "grp_R2-4-015", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
 ?et=core.rd-group&ep=grp_R2-4-015

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
    base="coap://[ff05::5:1]"
```

Similarly, to retrieve the multicast IP address used in "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
 ?et=core.rd-group&ep=grp_schedule

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/502>;ep="grp_schedule";et="core.rd-group";
    base="coap://[ff05::5:2]"
```

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

Having learnt the application groups to which the device belongs, the device learns the security groups to which it belongs. In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
 ?rt=core.osc.mbr&app-gp=grp_R2-4-015

Response: RD -> Joining Node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/group-oscore/feedca570000>;
  rt="core.osc.mbr";sec-gp="feedca570000";
  app-gp="grp_R2-4-015";anchor="coap://[2001:db8::ab]"
```

Similarly, the device does the following for app-gp="grp_schedule".

Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
?rt=core.osc.mbr&app-gp=grp_schedule

Response: RD -> Joining Node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/group-oscore/feedsc590000>;
  rt="core.osc.mbr";sec-gp="feedsc590000";
  app-gp="grp_schedule";anchor="coap://[2001:db8::ab]"
```

*** *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g. according to [I-D.ietf-ace-key-groupcomm-oscore].

7. Security Considerations

The security considerations as described in Section 8 of [I-D.ietf-core-resource-directory] apply here as well.

8. IANA Considerations

This document has the following actions for IANA.

8.1. Resource Types

IANA is asked to enter the following value into the Resource Type (rt=) Link Target Attribute Values subregistry within the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690].

Value	Description	Reference
core.osc.mbr	Group-membership resource of an OSCORE Group Manager	[[this document]]

9. References

9.1. Normative References

- [I-D.ietf-ace-cwt-proof-of-possession]
 Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-11 (work in progress), October 2019.
- [I-D.ietf-ace-key-groupcomm-oscore]
 Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-03 (work in progress), November 2019.
- [I-D.ietf-core-oscore-groupcomm]
 Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-06 (work in progress), November 2019.
- [I-D.ietf-core-resource-directory]
 Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [Fairhair] FairHair Alliance, "Security Architecture for the Internet of Things (IoT) in Commercial Buildings", White Paper, ed. Piotr Polak , March 2018, <https://www.fairhair-alliance.org/data/downloadables/1/9/fairhair_security_wp_march-2018.pdf>.
- [I-D.dijk-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-01 (work in progress), July 2019.
- [I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-25 (work in progress), October 2019.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [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/info/rfc8132>>.

[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/info/rfc8613>>.

Acknowledgments

The authors sincerely thank Carsten Bormann, Klaus Hartke, Francesca Palombini, Dave Robin and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC, and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Peter van der Stok
Consultant

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