

CoRE  
Internet-Draft  
Intended status: Informational  
Expires: September 12, 2019

C. Amsuess  
March 11, 2019

Resource Directory Replication  
draft-amsuess-core-rd-replication-02

Abstract

Discovery of endpoints and resources in M2M applications over large networks is enabled by Resource Directories, but no special consideration has been given to how such directories can scale beyond what can be managed by a single device.

This document explores different ways in which Resource Directories can be scaled up from single network to enterprise and global scale. It does not attempt to standardize any of those methods, but only to demonstrate the feasibility of such extensions and to provide terminology and exploratory groundwork for later documents.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Goals of upscaling . . . . .	3
3.1. Large numbers of registrations . . . . .	3
3.2. Large number of requests . . . . .	3
3.3. Redundancy . . . . .	4
4. Approaches . . . . .	4
4.1. Shared authority . . . . .	4
4.2. Plain caching . . . . .	5
4.3. RD-aware caching . . . . .	6
4.3.1. Potential for improvement . . . . .	7
4.4. Distinct registration points . . . . .	7
4.4.1. Redundancy and handover . . . . .	8
4.4.2. Loops between RDs and proxies . . . . .	8
5. Proposed RD extensions . . . . .	9
5.1. Provenance . . . . .	9
5.2. Lifetime reporting . . . . .	10
6. Example scenarios . . . . .	11
6.1. Redundant and replicated resource lookup (anycast) . . . . .	11
6.2. Redundant and replicated resource lookup (distinct registration points) . . . . .	12
6.2.1. Variation: Large number of registrations, localized queries . . . . .	15
6.2.2. Variation: Combination with anycast . . . . .	15
6.3. Anonymous global endpoint lookup . . . . .	16
7. References . . . . .	18
7.1. Informative References . . . . .	18
7.2. URIs . . . . .	19
Author's Address . . . . .	19

## 1. Introduction

[ See abstract for now. ]

This document is being developed in a git based workflow. Please see <https://github.com/chrysn/resource-directory-replication> [1] for more details and easy ways to contribute.

## 2. Terminology

This document assumes familiarity with [RFC7252] and [I-D.ietf-core-resource-directory] and uses terminology from those documents.

Examples in which URI paths like `"/rd"` or `"/rd-lookup/res"` are used assume that those URIs have been obtained before by an RD Discovery process; these paths are only examples, and no implementation should make assumptions based on the literal paths.

## 3. Goals of upscaling

The following sections outline different reasons why a Resource Directory should be scaled beyond a single device. Not all of them will necessarily apply to all use cases, and not all solution approaches might be suitable for all goals.

### 3.1. Large numbers of registrations

Even at 1kB of link data per registration, modern server hardware can easily keep the data of millions of registrations in RAM simultaneously. Thus, the mere size of registration data is not expected to be a factor that requires scaling to multiple nodes.

The traffic produced when millions of nodes with the default 24h lifetime amounts to dozens of exchanges per second, which is doable with equal ease at central network equipment.

However, if the directory has additional interaction with its registered nodes, for example because it provides proxying to registered endpoints, resources like file descriptors can be exhausted earlier, and the traffic load on the registration server grows with the traffic it is proxying for the endpoint.

### 3.2. Large number of requests

Not all approaches to constrained restful communication use the Resource Directory only in the setup stage; some might also utilize a Resource Directory in more day-to-day operation.

[ TODO: get some numbers on how many requests a single RD can deal with. ]

### 3.3. Redundancy

With the RD as a central part of CoRE infrastructures, outages can affect a large number of users.

A decentralized RD should be able to deal both with scheduled downtimes of hosts as well as unexpected outages of hosts or parts of the network, especially with network splits between the individual parts of the directory.

## 4. Approaches

In this section, two independent chains of approaches are presented. The "shared authority" approach (using anycast or DNS aliases), and proxy-based caching (in stages from using generic proxies to RD replication that only bears little resemblance to proxies).

In the remainder of this document, the term "proxy" always refers to a device which a client can access as if it were a resource directory, and forwards the request to an actual RD.

Elements from those chains can be mixed.

### 4.1. Shared authority

With this approach, a single host and port (or "authority" component in the generic URI syntax) is used for all interactions with the RD.

This can be implemented using a host name pointing to different IP addresses simultaneously or depending on the requester's location, using IP anycast addresses or both.

From the client's or proxy's point of view, all interaction happens with same Origin Server.

In this setup, the replication is hidden from the REST interactions, and takes place inside the RD server implementation or its database backend.

Compared to the other approaches, this is more complex to set up when it involves managing anycast addresses: Running an IPv4 anycast network on Internet scale requires running an Autonomous System. In either variation, all server instances are tightly coupled; they need shared administration and probably need to run the same software.

The replication characteristics are largely inherited from the underlying backend.

As registering endpoints only store the URI constructed from the Location-Path option to their registration request, registration updates can end up at any instance of the server, though they are likely to reach the same one as before most of the time.

Spontaneous failure of individual nodes can interrupt endpoints' registrations in scenarios that do not use anycast addresses until the unusable addresses have left DNS caches.

#### 4.2. Plain caching

Caching reverse proxies that are not particularly aware of a Resource Directory can be used to mitigate the effect of large numbers of requests on a single RD server. In this approach, there exists a single central RD server instance, but proxies are placed in front of it to reduce its load.

Caching is applicable only to the lookup interfaces; the POST request used in registration and renewal are not cacheable.

A prerequisite for successful caching is that fresh copies exist in the cache; this is likely to happen only if there are many alike requests to the Resource Directory. The proxy can then serve cached copies, and might find it advantageous to observe frequent queries.

The simplest way to set up such proxying is to have the proxies forward all requests to the central RD and to advertise only the proxies' addresses.

Due to the discovery process of the RD, operators can also limit the proxies to the lookup interfaces and advertise the central server for registration purposes. A sample exchange between a node and its 6LoWPAN border router could be:

```
Req: GET coap://[fe80::1]/.well-known/core?rt=core.rd*
```

```
Res: 2.05 Content
```

```
<coap://central-rd.example.com/rd>;rt="core.rd",  
<coap://europe3.proxy.rd.example.com/rd-lookup/res>;rt="core.rd-lookup-res",  
<coap://europe3.proxy.rd.example.com/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

Special care should be taken when a reverse proxy is not accessed by the client under the same address as the origin server, as relative references change their meaning when served from there. This can be ignored completely on the resource lookup interface (as long as the provenance extension is not used); ignoring it on the endpoint lookup interface gives the client "wrong" results, though that is likely to only matter to applications that use both the lookup and the

registration interface, like Commissioning Tools could do. Proxies can be configured to do content transcoding (cf. [RFC8075] Section 6.5.2) to preserve the lookup responses' original meanings.

This approach does not help at all with large numbers of registrations. It can mitigate issues with large numbers of lookup requests, provided that many identical requests arrive at the proxy. The effect on the redundancy goal is negligible: The proxy can provide lookup results only for as long as the cache is fresh during a central server outage, which is 60 seconds unless the RD server says otherwise.

This approach can be run with off-the-shelf RD servers and proxies. The only configuration required is for the proxy to have a forwarding address, and for the RD (or its announcer) to know which lookup addresses to advertise.

#### 4.3. RD-aware caching

Similar to the above, specialized proxies can be employed that are aware that their target is an RD lookup address.

The "plain caching" approach is limited in that it requires a small set of lookups to be frequently performed. A proxy that is aware that the address it is forwarding to is of the Resource Type "core.rd-lookup-\*" can utilize knowledge of how an RD works to serve more specialized requests as well from fresh generic content.

For example, assume that the proxy frequently receives requests of the shape

```
Req: GET /rd-lookup/res?rt=core.s&rt=ex.temperature&ex.building=8341&title=X
```

for arbitrary values of X. Then it can use the following request to keep a fresh cache:

```
Req: GET coap://rd.example.com/rd-lookup/res?rt=core.s&rt=ex.temperature
      &ex.building=8341
Observe: 1
```

and from that serve filtered responses to individual requests.

This method shares the advantages of plain caching, with reduced limitations but requiring specialized proxying software. The software does not necessarily need more configuration: A general-purpose proxy is free to explore the origin server's ".well-known/core" information, and can decide to enable RD optimizations after

discovering that the frequently accesses resources are of resource type "core.rd-lookup-\*".

#### 4.3.1. Potential for improvement

Observing a large lookup result is relatively inefficient as the complete document needs to be transferred when a change happens. Serializations of web links that are suitable for expressing small deltas are expected to be developed for PATCH operations on registration resources. If those formats are compatible with observation, they can be applied directly. Otherwise, the proxy can try to establish a "push" dynamic link ([I-D.ietf-core-dynlink]) to receive continuous PATCH updates on its resource.

The applicability of the RD-aware approach is further limited to query parameters of which the proxy knows that they are not subject to lookup filtering on other entities than the queried one. In the example above, were the variable part the "d" attribute (of endpoints, as opposed to the "title" of resources), the proxy could not do the filtering on its own because it would not have the required information. Even the above example does not allow for fully accurate replication, as the endpoint `_might_` register with a "title" endpoint attribute, even though no such attribute is specified right now. Also, annotating the links in the endpoint lookup with information about which registration they belong to would help the proxy keep all the data around to solve more complex queries. The provenance extension is proposed for that purpose.

In its extreme form, the proxy can observe the complete endpoint lookup resource of the Resource Directory. and run a dedicated observation for each registration. It can then answer all queries on its own based on the continuously fresh state transferred in the observations.

For such proxies, it can be suitable to configure them to use stale cache values for extended periods of time when the RD becomes intermittently unavailable.

#### 4.4. Distinct registration points

Caching proxies that are aware of RD semantics could be extended to gather information from more than one Resource Directory.

When executing queries, they would consider candidates from all configured upstream servers and report the union of the respective query results. At this stage, it is highly recommended that content transcoding takes place.

With this approach, many distinct registration URIs can be advertised, for example due to geographic proximity.

Unlike the other proxying approaches, this helps with the "large number of registrations" goal. If that number is unmanageable for single devices, proxies need not keep full copies of all the RDs' states but rather send out queries to all of their upstreams, behaving more like the "plain caching" proxies. This multiplies the lookup traffic, but allows for huge numbers of registrations. The problems of "too many lookups" versus "too many registrations" can be traded off against each other if the proxies keep parts of the RDs' states locally at hand while forwarding more exotic requests to all RDs.

#### 4.4.1. Redundancy and handover

This approach also tackles the redundancy goal. When an endpoint registers at its RD, the RD updates its endpoint and resource lookup results and includes the registration data until further notice (for correct operation, the "Lifetime Age" extension is useful).

If at some point in time that RD server becomes unavailable, the proxies can keep the cached information around. Before the lifetime expires, the endpoint will attempt to renew its registration and find that the RD is unavailable. It will then go through discovery again, find the most recently advertised registration URI or pick another one out of a set and start a new registration there.

If the lookup proxies do not evict the old (and soon-to-time-out) registration when the new one on a different RD with the same endpoint name and domain arrives, at worst there will be the same information twice from two registration resources available for lookup.

#### 4.4.2. Loops between RDs and proxies

In this configuration, it can be tempting to run a Resource Directory and a lookup proxy (aimed at multiple resource directories) on the same host.

[ It might be easier to recommend simply using different hosts, at least host names, in those cases, or anything else that allows direct and not publically advertised access to the real RDs' lookups. ]

In such a setup, other aggregating lookup proxies must take care to only select locally registered entries. With the current filtering rules, observing the resources `"/rd-lookup/ep?href=/"` and `"/rd-lookup/res?provenance=/"` crudely provides that kind of filtering.



## 5. Proposed RD extensions

### 5.1. Provenance

In order for an RD-aware proxy to serve resource lookup requests that filter on endpoint parameters, the proxy needs a way to tell which endpoint registration submitted that link. That information might also be useful for other purposes.

This introduces a new link attribute "provenance". Its value is a URI reference as described by [RFC3986] Section 4.1. The URI is to be interpreted by the same rules that apply to the "anchor" attribute, namely by resolving the reference relative to the requested document's URI. The attribute should not be repeated, and in presence of multiple attributes, only the last should be considered.

[ TODO: If a something link-format-ish comes up during the development of this document which allows setting base-hrefs in-line, evaluate whether it really makes sense to inherit anchor's rules or whether it's better to phrase it in a way that the requested base URI always counts. A composite CoRAL endpoint-and-resource lookup on the RD might make this extension proposal obsolete. ]

The URI given in the "provenance" attribute describes where the information in the link was obtained from. An aggregator of links can thus declare its sources for each link.

It is recommended that a Resource Directory adds the URI of the registration resource to resource lookups. Thus, if an endpoint registers as

```
Req: POST /rd?ep=node1
Payload:
</sensors/temp>;if="core.s"
```

```
Res: 2.01 Created
Location: /reg/1234
```

then a lookup will add a provenance attribute:

```
Req: GET /rd-lookup/res?if=core.s

Res: 2.05 Content
Payload:
<coap://.../sensors/temp>;if="core.s";anchor="coap://...";
  provenance="/reg/1234"
```

This is not an IANA consideration as there is no established registry of link attributes.

By itself, the provenance attribute does not need to be registered in the RD Parameters Registry because it is just another link attribute. If it is desired that provenance information is only shown on request (eg. by RD-aware proxies), a parameter can be introduced there:

- o Full name: Link provenance
- o short: provenance
- o Validity: URI
- o Use: Resource lookup only
- o Description: If "provenance" or any string starting with "provenance=" is given as one of the ampersand-delimited query arguments, the RD is instructed to add the provenance attribute to all looked up links; otherwise, the RD will not present them. The filtering rules still apply: If there is a "=" sign in the query argument, only links with matching provenance will be reported.

## 5.2. Lifetime reporting

The result of an endpoint lookup as a whole has inhomogenous cache properties that would determine its Max-Age:

- o The document can change at any time when a new endpoint registers.
- o The document can change at any time when an endpoint deregisters.
- o Each record can be expected to not change until its lifetime has expired.

As currently specified, a lookup client has no way to tell where in its lifetime an endpoint is. Therefore, a new link attribute is suggested that allows the RD to share that information:

The new link attribute Lifetime Remaining (lt-remaining) is described for use in RD Endpoint Lookups. Valid values are integers from 0 to the lifetime of the registration. The value indicates how many seconds have passed since the endpoint last renewed its registration.

Care has to be taken when replicating this value in caches, as the caching agent might be unaware of the attribute's semantics and not update it. (This is unlike the Max-Age attribute, which a caching agent needs to understand and reduce accordingly when serving from

the cache). It should therefore only be used with responses that carry the default Max-Age of 60 or less.

Clients that use the lookup interface (especially RD-aware proxies) are free to treat that record and its corresponding resource records as fresh until after *lt*-remaining seconds have passed since the endpoint lookup result was obtained, especially if the origin server has become unavailable.

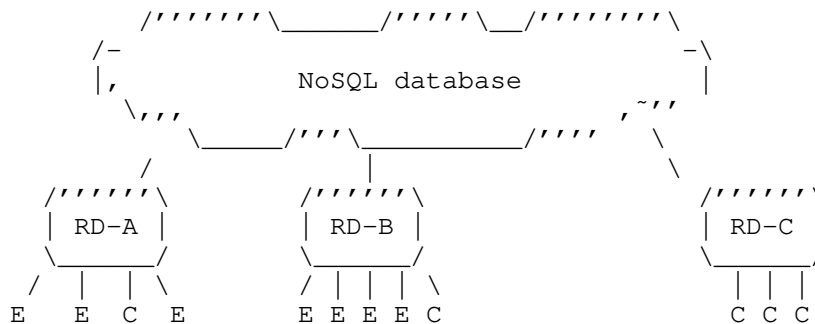
**Security considerations:** Given that this leaks information about the endpoint's communication patterns, it may be prudent for an RD only to reveal this information on a need-to-know basis.

## 6. Example scenarios

### 6.1. Redundant and replicated resource lookup (anycast)

This scenario describes a setup where millions of devices register in a company-wide Resource Directory.

The directory is scaled using the shared authority / anycast approach, and the RD implementation is backed by a NoSQL-style distributed database.



("E" and "C" represent endpoints and lookup clients, respectively)

Both endpoints and lookup clients receive the RD address "2001:db8::an1:ca57" is announced to all devices on the network using the RDAO option in IPv6 Neighbor Discovery. Any packages to that addresses are routed by the network to the closest of the three RD instances A, B and C. Discovery invariably looks like this:

```
Req: GET coap://[2001:db8::an1:ca57]/.well-known/core?rt=core.rd*
```

```
Res: 2.05 Content
</rd>;rt="core.rd",
</rd-lookup/res>;rt="core.rd-lookup-res",
</rd-lookup/ep>;rt="core.rd-lookup-ep"
```

An endpoint close to B would therefore register with

```
Req: POST coap://[2001:db8::an1:ca57]/rd?ep=endpoint1&
      d=facility23.eu.example.com
Payload:
</sensors/temp>;if="core.s"
```

```
Res: 2.01 Created
Location: /reg/123e4567-e89b-12d3-a456-426655440000
```

Any client could immediately see that the endpoint is registered by issuing

```
Req: GET coap://[2001:db8::an1:ca57]/rd-lookup/ep?ep=endpoint1&
      d=facility23.eu.example.com
```

```
Res: 2.05 Content
Payload:
</reg/123e4567-e89b-12d3-a456-426655440000>;ep="endpoint1";
      d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"
```

If at any point in time the RD instance B becomes unavailable, the registering endpoint's renewal requests will be routed to the next available instance, for example A. That instance can update the shared database with renewed lifetime just as B would have done.

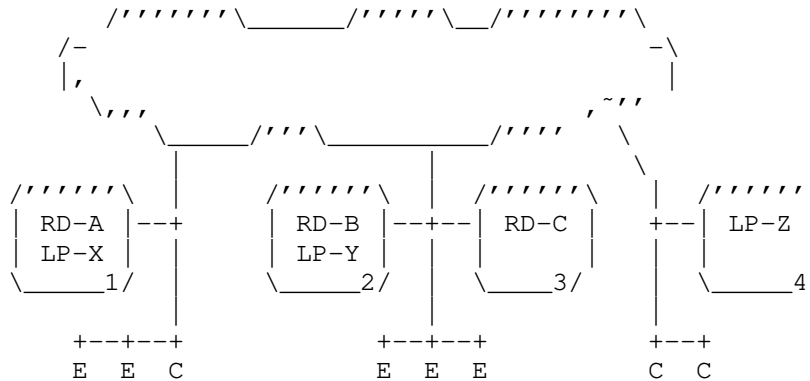
How this performs under a net split depends on the database backend. Registration resources based on UUIDs were chosen in this example because those would allow the system to keep accepting new registrations even in a netsplit situation; the risk of the registration request not being idempotent towards a node that switches sides during such a split is considered acceptable.

## 6.2. Redundant and replicated resource lookup (distinct registration points)

This scenario takes place in the same environment as the previous one.

Rather than a shared database, distinct registration points are advertised. The advertised registration points are called RD-A to

RD-C; independent of them are lookup proxies LP-X to LP-Z. Some of them run on the same hosts.



The lookup proxies in this scenario are constantly observing the `"/rd-lookup/ep?href=/*"` and `"/rd-lookup/res?provenance=/*"` resources of known RDs on other hosts, and might get updated internally with state from a co-hosted RD or observe that using an internal interface. As there is no really suitable content format and observation mechanism for those yet, the exchanges are partially described in words here.

RDAO announcements point to the nearest host (whose IP address ends with the numbers of the respective box in the figure), and hosts that do not serve both functions provide lookup as follows:

```
Req: GET coap://[2001:db8:23::3]/.well-known/core?rt=core.rd*
```

```
Res: 2.05 Content
```

```
Payload:
```

```
</rd>;rt="core.rd",
```

```
<coap://[2001:db8:23::2]/rd-lookup/ep>;rt="core.rd-lookup-ep",
```

```
<coap://[2001:db8:23::2]/rd-lookup/res>;rt="core.rd-lookup-res"
```

When a client then registers as

```
Req: POST coap://[2001:db8:23::3]/rd?ep=endpoint1&
      d=facility23.eu.example.com
```

```
Payload:
```

```
</sensors/temp>;if="core.s"
```

```
Res: 2.01 Created
```

```
Location: /reg/42
```

the RD at 3 sends notifications to the observing lookup proxies X, Y and Z:

Res: Patch Result

Add one record: </reg/42>;ep="endpoint1";d="facility23.eu.example.com";  
con="coap://[2001:db8:23::1]";lt-remaining=90000

As soon as that is processed, clients can query LP-Z

Req: GET coap://[2001:db8:4::4]/rd-lookup/ep?ep=endpoint1&  
d=facility23.eu.example.com

Res: 2.05 Content

Payload:

<coap://[2001:db8:23::3]/reg/42>;ep="endpoint1";  
d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"

(Note that lt-remaining is elided to the client as per the security considerations for that information).

When a net split happens that cuts LP-Z's site off the rest, it keeps that information available until the lt-remaining runs out.

When RD-C unexpectedly becomes unavailable, endpoint1 fails to renew its registration. It then starts the RD discovery process again, picks the next available RD (this time B) and gets a new registration from that.

RD-B then sends an update to the proxies:

Res: Patch Result

Add one record: </reg/11>;ep="endpoint1";d="facility23.eu.example.com";  
con="coap://[2001:db8:23::1]";lt-remaining=90000

The proxies remove C's registration "/reg/42" based on the duplicate name and now answer requests like this:

Req: GET /rd-lookup/ep?ep=endpoint1&d=facility23.eu.example.com

Res: 2.05 Content

Payload:

```
<coap://[2001:db8:23::2]/reg/11>;ep="endpoint1";  
    d="facility23.eu.example.com";con="coap://[2001:db8:23::1]"
```

Req: GET /rd-lookup/res?if=core.s&d=facility23.eu.example.com

Res: 2.05 Content

Payload:

```
<coap://[2001:db8:23::1]/sensors/temp>;if="core.s";  
    anchor="coap://[2001:db8:23::1]/sensors/temp";  
    provenance="coap://[2001:db8:23:2]/reg/11",  
    ...
```

#### 6.2.1. Variation: Large number of registrations, localized queries

If the lookup proxies are not capable of keeping track of all the registered data, they can opt to forward requests to all the RDs instead. In this example, queries are often localized (queries within a building are often limited to the same building), so LP-Y could decide to only keep two particular observations active to each RD:

- o "/rd-lookup/ep?href=/\*&d=facility23.eu.example.com"

- o "/rd-lookup/res?provenance=/\*&d=facility23.eu.example.com"

With those observed, it could still accurately respond to the above queries without calling out to the other RDs.

If a query came in as "/rd-lookup/res?if=core.s", it would still need to forward that query to all RDs to build an overview of all sensors in the network for the requester.

#### 6.2.2. Variation: Combination with anycast

In a variation of this, all the RDs and LPs can use a shared anycast address. They would be then advertised as in the anycast/NoSQL example.

All RDs would need to be configured such that they encode their host name in their path (eg. "/reg/rd-c/42"). Nodes must then have proxy forwarding rules set up such that

- o "/rd" is served from the local RD if there is one, or forwarded to any (the closest) RD

- o `"/reg/*"` requests are served if hosted locally, otherwise forwarded to the appropriate RD, or respond with a 5.04 Gateway timeout if that is not available any more
- o Lookup request are served from the local lookup proxy, or forwarded to the closest one on RD-only hosts.

Such a setup is easier if all hosts provide both registration and lookup functionality.

### 6.3. Anonymous global endpoint lookup

This scenario describes a way to provide connectivity into devices in difficult network situations based on identifiers of their cryptographic keys, in this case the (sufficiently long) ID context plus recipient ID of OSCORE ([I-D.ietf-core-object-security]). A global network of untrusted Resource Directory servers is built, and the individual servers provide network relaying services for endpoints that operate behind NAT or firewalls.

It assumes the existence of two other hypothetical mechanisms:

- o The `"proxy"` parameter from [I-D.amsuess-core-resource-directory-extensions]
- o A URI scheme called `"oscore"`.

A URI of the form `"oscore://VGhh...2aWNl/sensor/temp"` refers to a resource `"/sensor/temp/"` on any OSCORE capable host with which the client has a key established under the KID context and recipient ID given by the base64 string in the authority component.

To resolve the URI to a concrete protocol and socket, a form of Resource Directory assisted protocol negotiation is used.

RD servers join a global pool of servers using a protocol that is not further described here, but could conceivably be based on distributed hash tables (DHTs).

Endpoints register only with a key derived name, and usually do not provide any links because those would be accessible only to authenticated requesters.

They register at any of a set of preconfigured DNS names for finding a Resource Directory. Those names resolve to any of the currently active RD servers, where geographic proximity could play a role in the choice of address returned.



When the endpoint discovers the registration URI (for which it uses coap+tcp to make later proxying more stable), the server returns links to its explicit IP address:

```
<coap+tcp://[2001:db8:1:2::3]/rd>;rt="core.rd",  
<coap+tcp://[2001:db8:1:2::3]/rd-lookup/ep>;rt="core.rd-lookup-ep"
```

(This avoids conflict when the DNS assignment flips and a different host (on which the registration resource is unknown) is returned. Alternatively, the servers could use a unified scheme of registration resource naming like "/reg/\${name}" or a UUID-based scheme.)

The endpoint then registers:

```
Req: POST coap+tcp://[2001:db8:1:2::3]/rd?proxy&ep=VGhhdCdzIHRobzSB\  
      LZx1JZENvbnRleHQgdXNlZCB3aXRoIHRobXMgZGV2aWNl  
Payload: empty
```

```
Res: 2.01 Created  
Location: /reg/123
```

When a client wants to talk to that registered server, its RD discovery process will yield another instance, which it then queries:

```
Req: GET coap://[2001:db8:4:5::6]/rd-lookup/ep?ep=VGhhdCdzIHRobzSBL\  
      ZXlJZENvbnRleHQgdXNlZCB3aXRoIHRobXMgZGV2aWNl
```

The server will look up the given ep name in the backing DHT, and forward the request right to the (precisely: any) RD server that has announced that ep value, which then answers:

```
Res: 2.05 Created  
Payload:  
<coap+tcp://[2001:db8:1:2::3]/reg/123>;ep="VGhh...2aWNl";  
  con="coap://[2001:db8:1:2::3]:10123";  
  at="coap+tcp://[2001:db8:1:2::3]:10123"
```

(This particular server uses multiple ports to tell traffic for different endpoints apart; it could just as well use a catch-all DNS record, do name based virtual hosting and announce "con="coap://reg123.server3.example.com" instead.)

The client will then use the discovered address to direct its OSCORE requests to, and the RD server will proxy for it.

Note that while this setup can serve as a generic RD and answer resource requests as well, it is doubtful whether there would be any interest in it given the data becomes public, and is limited by the

necessity to have an "ep=" filter in all requests lest the network be flooded with requests.

## 7. References

### 7.1. Informative References

- [I-D.amsuess-core-resource-directory-extensions]  
Amsuess, C., "CoRE Resource Directory Extensions", draft-amsuess-core-resource-directory-extensions-00 (work in progress), January 2019.
- [I-D.ietf-core-dynlink]  
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-08 (work in progress), March 2019.
- [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-19 (work in progress), January 2019.
- [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>>.
- [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>>.

## 7.2. URIs

[1] <https://github.com/chrysn/resource-directory-replication>

### Author's Address

Christian Amsuess  
Hollandstr. 12/4  
1020  
Austria

Phone: +43-664-9790639  
Email: christian@amsuess.com

CoRE  
Internet-Draft  
Intended status: Experimental  
Expires: July 14, 2019

C. Amsuess  
January 10, 2019

CoRE Resource Directory Extensions  
draft-amsuess-core-resource-directory-extensions-00

Abstract

[ See Introduction ]

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 July 14, 2019.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	2
2. Reverse Proxy requests . . . . .	2
2.1. Discovery . . . . .	3
2.2. Registration . . . . .	3
2.2.1. Registration updates . . . . .	4
2.2.2. On-Demand proxying . . . . .	4
2.2.3. Examples . . . . .	4
2.2.4. Notes on stability and maturity . . . . .	6
2.2.5. Security considerations . . . . .	6
3. Infinite lifetime . . . . .	6
3.1. Example . . . . .	6
4. Lookup across link relations . . . . .	7
4.1. Example . . . . .	7
5. Lifetime Age . . . . .	8
6. Zone identifier introspection . . . . .	8
6.1. Example . . . . .	8
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	9
7.3. URIs . . . . .	9
Author's Address . . . . .	10

## 1. Introduction

This document pools some extensions to the Resource Directory [I-D.ietf-core-resource-directory] that might be useful but have no place in the original document.

They might become individual documents for IETF submission, simple registrations in the RD Parameter Registry at IANA, or grow into a shape where they can be submitted as a collection of tools.

At its current state, this draft is a collection of ideas.

[ This document is being developed at <https://gitlab.com/chrysn/resource-directory-extensions> [1]. ]

## 2. Reverse Proxy requests

When a registrant registers at a Resource Directory, it might not have a suitable address it can use as a base address. Typical reasons include being inside a NAT without control over port forwarding, or only being able to open outgoing connections (as program running inside a web browser utilizing CoAP over WebSocket [RFC8323] might be).

[I-D.ietf-core-resource-directory] suggests (in the Cellular M2M use case) that proxy access to such endpoints can be provided, it gives no concrete mechanism to do that; this is such a mechanism.

## 2.1. Discovery

An RD that provides proxying functionality advertizes it by announcing the additional resource type "TBD1" on its directory resource.

## 2.2. Registration

A client passes the "proxy=yes" or "proxy=ondemand" query parameter in addition to (but typically instead of) a "base" query parameter.

A server that receives a "proxy=yes" query parameter in a registration (or receives "proxy=ondemand" and decides it needs to proxy) MUST come up with a "Proxy URL" on which it will act as a reverse proxy for the registrant and which it uses as a Registration Base URI for the present registration.

The Proxy URL SHOULD have no path component, as acting as a reverse proxy in such a scenario means that any relative references in all representations that are proxied must be recognized and possibly rewritten.

The RD MAY mint several alternative Registration Base URIs using different protocols to make the proxied content available; [I-D.silverajan-core-coap-protocol-negotiation] can be used to advertise them.

The registrant is not informed of the chosen public name by the RD.

If an explicit "base" parameter is given, the RD will forward requests to the Proxy URL to that location. Otherwise, it forwards to the registration's source address (which is the implied base parameter).

This mechanism is applicable to all transports that can be used to register. If proxying is active, the restrictions on when the base parameter needs to be present ([I-D.ietf-core-resource-directory] Registration template) are relaxed: The base parameter may also be absent if the connection originates from an ephemeral port, as long as the underlying protocol supports role reversal, and link-local IPv6 addresses may be used without any concerns of expressibility.

If the client uses the role reversal rule relaxation, it keeps that connection open for as long as it wants to be reachable. When the connection terminates, the RD SHOULD treat the registration as having

timed out (even if its lifetime has not been exceeded) and MAY eventually remove the registration.

#### 2.2.1. Registration updates

The "proxy" query parameter can not be changed or repeated in a registration update; RD servers MUST answer 4.00 Bad Request to any registration update that has a "proxy" query parameter.

As always, registration updates can explicitly or implicitly update the Registration Base URI. In proxied registrations, those changes are not propagated to lookup, but do change the forwarding address of the proxy.

For example, if a registration is established over TCP, an update can come along in a new TCP connection. Starting then, proxied requests are forwarded along that new connection.

Note that transports can not be switched in a registration update, as the protocol is part of the registration resource.

#### 2.2.2. On-Demand proxying

If an endpoint is deployed in an unknown network, it might not know whether it is behind a NAT that would require it to configure an explicit base address, and ask the RD to assist by proxying if necessary by registering with the "proxy=ondemand" query parameter.

A server receiving that SHOULD use a different source port to try to access the registrant's .well-known/core file using a GET request under the Registration Base URI. If that succeeds, it may assume that no NAT is present, and ignore the proxying request. Otherwise, it configures proxying as if "proxy=yes" were requested.

Note that this is only a heuristic [ and not tested in deployments yet ].

#### 2.2.3. Examples

##### 2.2.3.1. Registration through a firewall

Req from [2001:db8:42::9876]:5683:  
POST coap://rd.example.net/rd?ep=node9876&proxy=ondemand  
</some-resource>;rt="example.x"

Req from rd.example.net:49152:  
GET coap://[2001:db8:42::9876]/.well-known/core

Request blocked by stateful firewall around [2001:db8:42::]

RD decides that proxying is necessary

Res: 2.04 Created  
Location: /reg/abcd

Later, lookup of that registration might say:

Req: GET coap://rd.example.net/lookup/res?rt=example.x

Res: 2.05 Content  
<coap://node987.rd.example.net/some-resource>;rt="example.x"

A request to that resource will end up at an IP address of the RD, which will forward it using its the IP and port on which the registrant had registered as source port, thus reaching the registrant through the stateful firewall.

#### 2.2.3.2. Registration from a browser context

Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes  
</gyroscope>;rt="core.s"

Res: 2.04 Created  
Location: /reg/123

The gyroscope can now not only be looked up in the RD, but also be reached:

Req: GET coap://rd.example.net/lookup/res?rt=core.s

Res: 2.05 Content  
<coap://[2001:db8:1::1]:10123/gyroscope>;rt="core.s"

In this example, the RD has chosen to do port-based rather than host-based virtual hosting and announces its literal IP address as that allows clients to not send the lengthy Uri-Host option with all requests.



#### 2.2.4. Notes on stability and maturity

Using this with UDP can be quite fragile; the author only draws on own experience that this can work across cell-phone NATs and does not claim that this will work over generic firewalls.

[ It may make sense to have the example as TCP right away. ]

#### 2.2.5. Security considerations

An RD MAY impose additional restrictions on which endpoints can register for proxying, and thus respond 4.01 Unauthorized to request that would pass had they not requested proxying.

Attackers could do third party registrations with an attacked device's address as base URI, though the RD would probably not amplify any attacks in that case.

The RD MUST NOT reveal the address at which it reaches the registrant except for adequately authenticated and authorized debugging purposes, as that address could reveal sensitive location data the registrant may wish to hide by using a proxy.

Usual caveats for proxies apply.

### 3. Infinite lifetime

An RD can indicate support for infinite lifetimes by adding the resource type "TBD2" to its list of resource types.

A registrant that wishes to keep its registration alive indefinitely can set the lifetime value as "lt=inf".

Registrations with infinite lifetimes never time out.

Infinite lifetimes SHOULD only be used by commissioning tools, or for proxy registrations over stateful connections.

#### 3.1. Example

Had the example of Section 2.2.3.2 discovered support for infinite lifetimes during lookup like this:

```
Req: GET coaps+ws://rd.example.net/.well-known/coer?rt=core.rd*
```

```
Res: 2.05 Content
</rd>;rt="core.rd TBD1 TBD2";ct=40
```

it could register like that:

```
Req: POST coaps+ws://rd.example.net/rd?ep=node1234&proxy=yes&lt=inf
</gyroscope>;rt="core.s"
```

```
Res: 2.04 Created
Location: /reg/123
```

and never need to update the registration for as long as the websocket connection is open.

(When it gets terminated, it could try renewing the registration, but needs to be prepared for the RD to already have removed the original registration.)

#### 4. Lookup across link relations

Resource lookup occasionally needs execute multiple queries to follow links.

An RD server (or any other server that supports [RFC6690] compatible lookup), can announce support for following links in resource lookups by announcing support for the TBD3 interface type on its resource lookup.

A client can the query that server to not only provide the matched links, but also links that are reachable over relations given in "follow" query parameters.

##### 4.1. Example

Assume a node presents the following data in its `<.well-known/core>` resource (and submitted the same to the RD):

```
</temp>;if="core.s";rt="example.temperature",
</t-prot>;rel="calibration-protocol";anchor="/temp",
<http://vendor.example.com/temp9000>;rel="describedby";anchor="/temp",
</hum>;if="core.s";rt="example.humidity",
</h-prot>;rel="calibration-protocol";anchor="/hum",
```

A lookup client can, in one query, find the temperature sensor and its relevant metadata:

```
Req: GET /rd-lookup/res?rt=example.temperature&follow=calibration-protocol&follow=describedby
```

```
<coap://node1/temp>;if="core.s";rt="example.temperature";anchor="coap://node1",
<coap://node1/t-prot>;rel="calibration-protocol";anchor="coap://node1/temp",
<http://vendor.example.com/temp9000>;rel="describedby";anchor="coap://node1/temp",
```

[ There is a better example [2] in an earlier stage of  
[I-D.tiloca-core-oscore-discovery] ]

[ Given the likelihood of a CoRAL based successor to [RFC6690], this  
lookup variant might easily be superseded by a CoRAL FETCH format.  
]

## 5. Lifetime Age

This extension is described in [I-D.amsuess-core-rd-replication]  
Section 5.2.

The "provenance" extension in Section 5.1 of the same document should  
probably be expressed differently to avoid using non-target link  
attributes.

## 6. Zone identifier introspection

The 'split-horizon' mechanism introduced in  
[I-D.ietf-core-resource-directory] (-19) (that registrations with  
link-local bases can only be read from the zone they registered on)  
reduces the usability of the endpoint lookup interface for debugging  
purposes.

To allow an administrator to read out the "show-zone-id" query  
parameter for endpoint and resource lookup is introduced.

A Resource Directory that understands this parameter MUST NOT limit  
lookup results to registrations from the lookup's zone, and MUST use  
[RFC6874] zone identifiers to annotate which zone those registrations  
are valid on.

The RD MUST limit such requests to authenticated and authorized  
debugging requests, as registrants may rely on the RD to keep their  
presence secret from other links.

### 6.1. Example

Req: GET /rd-lookup/ep?show-zone-id&et=printer

Res: 2.05 Content

</reg/1>;base="coap://[2001:db8::1]";et=printer;ep="bigprinter",  
</reg/2>;base="coap://[fe80::99%wlan0]";et=printer;ep="localprinter-1234",  
</reg/3>;base="coap://[fe80::99%eth2]";et=printer;ep="localprinter-5678",

## 7. References

### 7.1. Normative References

- [I-D.amsuess-core-rd-replication]  
Amsuess, C., "Resource Directory Replication", draft-amsuess-core-rd-replication-01 (work in progress), March 2018.
- [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-18 (work in progress), December 2018.
- [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>>.

### 7.2. Informative References

- [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.
- [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-00 (work in progress), October 2018.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [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>>.

### 7.3. URIs

- [1] <https://gitlab.com/chrysn/resource-directory-extensions>
- [2] <https://github.com/ace-wg/ace-oauth/issues/120#issuecomment-407997786>

Author's Address

Christian Amsuess  
Hollandstr. 12/4  
1020  
Austria

Phone: +43-664-9790639  
Email: christian@amsuess.com

CoRE  
Internet Draft  
Intended status: Standards Track  
Expires: August 2019

A. Bhattacharyya  
S. Agrawal  
H. Rath  
A. Pal  
B. Purushothaman  
TATA CONSULTANCY SERVICES LTD.  
February 6, 2019

Adaptive RESTful Real-time Live Streaming for Things (A-REaLiST)  
draft-bhattacharyya-core-a-realist-02

Abstract

This draft presents extensions to Constrained Application Protocol (CoAP) to enable RESTful Real-time Live Streaming for improving the Quality of Experience (QoE) for delay-sensitive Internet of Things (IoT) applications. The overall architecture is termed "Adaptive RESTful Real-time Live Streaming for Things (A-REaLiST)". It is particularly designed for applications which rely on real-time augmented vision through live First Person View (FPV) feed from constrained remote agents like Unmanned Aerial Vehicle (UAV), etc. These extensions provide the necessary hooks to help solution designers ensure low-latency transfer of streams and, for contents like video, a quick recovery from freeze and corruption without incurring undue lag. A-REaLiST is an attempt to provide an integrated approach to maintain the balance amongst QoE, resource-efficiency and loss resilience. It provides the necessary hooks to optimize system performance by leveraging contextual intelligence inferred from instantaneous information segments in flight. These extensions equip CoAP with a standard for efficient RESTful streaming for Internet of Things (IoT) contrary to HTTP-streaming in conventional Internet.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on August 6, 2019.

#### Copyright Notice

Copyright (c) 2018 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.....	3
2. Revisiting CoAP.....	5
2.1. Some Interesting Aspects of CoAP.....	5
2.2. The Prevalent Approaches for Streaming over Internet.....	5
2.3. CoAP as the Best of Two Worlds.....	6
3. The Approach behind A-REaLiST.....	6

3.1. Optional Context Aware Semantic Switch.....	6
4. The Options Introduced.....	7
5. The Handshake and Exchange Semantics.....	8
5.1. Initial Negotiation.....	9
5.2. Renegotiation.....	11
6. Some Design Guidelines.....	13
6.1. Implicit Congestion Avoidance.....	13
6.2. Considerations for Consumer-side Rendering.....	13
6.3. Determining the segment size.....	14
7. IANA Considerations.....	14
8. Security Considerations.....	15
9. References.....	15
9.1. Normative References.....	15
9.2. Informative References.....	15

## 1. Introduction

IoT emerged to facilitate exchange of frequent-but-small sensory information amongst numerous constrained sensors [IOT-ISOC][RFC7452]. However, recent trends in industry and research community realize the importance of live visual data as important sensory information. There are many discourses available to support this observation [Murphy]. Live First Person View (FPV) from Unmanned Aerial Vehicles (UAV) and dumb robot terminals are being used for futuristic remote control and actuation applications for Augmented Reality (AR), Visual Simultaneous Localization and Mapping (VSLAM), UAV based surveillance, etc. Efficacy of these applications depends on resource-efficient, low-latency, yet high QoE transfer of the FPV over the Internet (or IP networks in general). Contrary to the traditional video streaming applications, the UAV-like end-points (henceforth referred as 'video producer') that capture and transmit the FPV are resource constrained devices. Moreover, the producer may work in a lossy environment marred with fluctuating radio connectivity and disruptions due network congestion.

The QoE considerations of the video rendering unit (henceforth referred as 'video consumer') for these applications are quite different from traditional applications. For example, in case of highly delay sensitive AR applications, a human brain may not tolerate a noticeable video freeze or delayed reception, which might have been overlooked for usual content delivery service like a YouTube video. Such delay may result in wrong actuation. For example, delayed FPV from a UAV may lead to wrong control commands leading to catastrophic consequences. In addition, the communication should be as light-weight as possible to optimize the usage of on-board computing and energy resources of the UAV. So, real-time video transmissions for IoT applications require special treatment



[Pereira]. However, as revealed through a detail analysis of the state-of-the-art in the next section, the existing solutions do not address such special requirements. This draft attempts to bridge this important gap by extending CoAP [RFC7252].

To realize its purpose, the A-REaLiST architecture relies on [RFC7967] and adds few new header options which, taken together, can be conceived to form a conceptual 'Stream' extension on CoAP (Fig. 1).

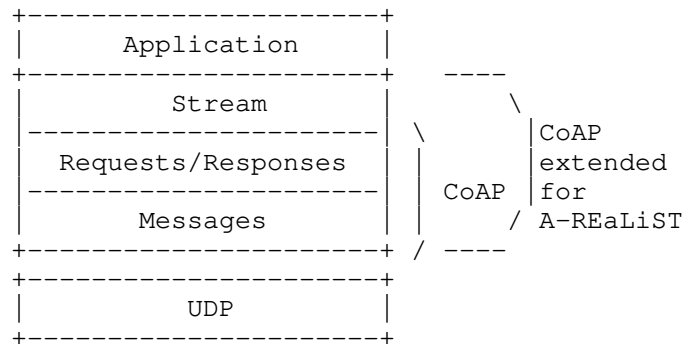


Figure 1: Abstract extended layering of CoAP for A-REaLiST with the conceptual layer for streaming.

Though primarily designed for video streaming, these extensions can also be used to allow streaming of time-series information on CoAP.

Note: Block-wise transfer [RFC7959] is a standardized extension to CoAP for transferring large application data. The cited use case for this is to perform firmware upgrade for a large number of constrained devices. Block-wise transfer is primarily concerned with reliable delivery of information. It works in synchronized manner. If a message remains unacknowledged despite retransmissions then the whole exchange is cancelled. So, it is not suitable for real-time delivery [GIoTS] which is requirement for many time-series information streams including video.

## 2. Revisiting CoAP

### 2.1. Some Interesting Aspects of CoAP

- ( i) CoAP allows both confirmable (CON) and non-confirmable (NON) messaging.
- ( ii) CON mode enables CoAP with an option for reliable RESTful delivery like HTTP [RFC2616] on TCP. On the other hand, intelligent use of No-Response option [RFC7967] along with NON mode can create an RTP like best-effort messaging on UDP.
- (iii) Context based switching between the reliable and best-effort semantics can be executed from the end-application level. This way an optimum balance between reliability delay-performance can be maintained to improve the overall Quality of Experience (QoE).
- ( iv) The base CoAP specification is inherently designed for resource constrained devices. Hence, a streaming protocol using the stateless RESTful semantics on CoAP makes the solution inherently lightweight. So, unlike conventional approach the designers can use a single stack that is equally efficient for sending the small data out of sensors, as well as, infinite visual stream.

### 2.2. The Prevalent Approaches for Streaming over Internet

The two prevalent approaches for streaming over the Internet are as below.

First approach is to send the information segment over HTTP which uses the reliability feature of the underlying Transmission Control Protocol (TCP) transport. In this case TCP state-machine puts more emphasis on reliable delivery of segments rather than maintaining the real-time deadlines. However, this is right now the prevalent approach as it treats video and other streams as general Internet traffic. So, streaming can seamlessly co-exist with the existing Internet architecture. Also, since TCP takes care of ordered delivery, the end-application does not need to worry about these matters.

The other approach is to use a specialized protocol like Real-time Transport Protocol (RTP) [RFC3550]. It treats video and other real-time streams as a special type of traffic. To ensure real-time delivery, the data is delivered in best-effort manner on top of UDP. So, reliable delivery is undermined.

### 2.3. CoAP as the Best of Two Worlds

It can be conjectured, tallying the above with previous section, that CoAP inherently imbibes the functional features from HTTP-on-TCP (reliable delivery) and RTP-on-UDP (best-effort delivery). Further CoAP allows the switching between these two seamlessly just by maneuvering the header options.

### 3. The Approach behind A-REaLiST

The design stems from the principles of "progressive download" on top of the RESTful request/response semantics of CoAP. The "producer" chunks the continuous information stream into segments as per the agreed maximum payload size suggested in [RFC7252]. Each chunk is transmitted as a CoAP request to a given resource at the "consumer". This draft provides the necessary header extensions that enable the "consumer" to maintain the sequence of the information segments in time and space.

#### 3.1. Optional Context Aware Semantic Switch

Before forming the CoAP message for each segment, the streaming application may use a real-time analytics module (henceforth referred as 'analytics module') which may provide inference to the "Stream" layer to decide the exchange semantics for the current segment. The message is sent reliably (CON message) or as best-effort (NON message with No-Response option) based on the segment's information criticality. Criticality is measured in terms of importance of the segment-content in reconstruction of the frames at the consumer. However, determination of criticality can be done on many aspects involving several application features like the source encoding type, the rendering logic at the consumer, etc. This way the over-all balance between QoE and resource-consumption may be maintained. Fig. 2 explains the idea with conceptual blocks. The overall concept and its efficacy has been explained with experimental results in [Wi-UAV-Globecom]

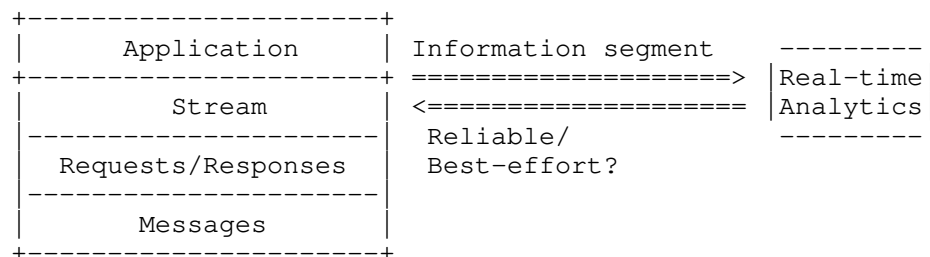


Figure 2: Illustrating the concept for context aware switching

Some examples are:

Example-1: Temporally compressed videos like MPEG consist of Group of Pictures (GoP) which comprises I-frames (Intra-frames) or key-frames, P-frames (Predicted frames) and B-frames (Bidirectional frames). Out of these 3 types of frames I-frames are most critical in terms of synchronizing with the GoP at the receiver end for successful rendering. So, an analytics module at the "video producer" end may infer each information segments of I-frames as critical and send those segments reliably. The segments corresponding to P and B frames may be transferred as best-effort requests.

Example-2: Let us consider a Motion JPEG (MJPEG) stream. In this case all the frames are independent JPEG frames and there is no temporal compression. The analytics module may treat the segments containing MJPEG meta-data for each frame as critical segments and transfer them through reliable messaging. Rest of the segments may be transferred as best-effort requests. An intelligent rendering engine at the "consumer" application may compensate for / conceal any possible loss of non-meta-data (non-critical) segments using the reliably received meta-data and rest of the non-meta-data segments received through best-effort. This way high QoE can be ensured despite reduced resource usage.

#### 4. The Options Introduced

To achieve the purpose of the Stream layer, three new protocol header options have been proposed as below:

- 1) Stream\_info: Consumes one unsigned byte. It maintains the stream identity and indicates the present phase of exchange. It is both a request and response option. It has two fields. The 3-LSBs indicate the state of exchange (Stream\_state) and 5-MSBs indicate an identifier (Stream\_id) for the stream. The identifier remains unchanged for the entire stream. So,

```
Stream_id = Stream_info >> 3;  
Stream_state = Stream_info & 0x7.
```

Interpretation of Stream\_state bits are :

000=> stream initiation (always with request);

001=> initiation accepted (always with response);

010=> initiation rejected (always with response);

011=> stream re-negotiation (with request or response);

100=> stream ongoing.

Note: While Stream\_id field enables to uniquely identify an information stream, it may also be used by an application to relate the context of different sub-streams (may be with varying QoS) and produce a combined rendering at the consumer-end.

- 2) Time-stamp: It consumes 32-bit unsigned integer. It is a request option. It relates a particular application information segment to the corresponding frame in the play sequence.
- 3) Position: It consumes 16-bit unsigned integer. It is a request option and MUST be accompanied with the Time-stamp option. It is a combination of two fields. The 15-MSBs indicate the "offset" at which the present segment is placed in the frame corresponding to the given timestamp. The LSB indicates if the current segment is the last segment of the frame corresponding to the given timestamp. Hence,  

$$\text{Last\_segment} = \text{Position} \ \&0x01 \ ? \ \text{True} : \ \text{False};$$

$$\text{Offset} = (\text{Position} \gg 1).$$

No.	C	U	N	R	Name	Format	Length	Default
TBD	X		-		Stream-info	uint	1	(none)
TBD	X		-		Time-stamp	uint	4	(none)
TBD	X		-		Position	uint	2	(none)

Table 1: Option Properties

## 5. The Handshake and Exchange Semantics

As per the design considerations (in view of the scenarios conceived at present) video transfer is initiated by the "producer" which acts as the client.

Each segment is transmitted to the "video consumer" as a POST request. The Time-stamp and Position options help sequential ordering of the segments at the consumer.

Note: The design considerations are driven by the experiences drawn from the applications where live video feeds are transmitted from battery operated constrained "video producers" like UAVs and dumb robotic terminals, etc. For example, while a fixed infrastructure system is using streamed FPV feed from UAVs, there may be situations where each time a UAV is low on resources (energy and computation, a new UAV with better state of resources (fresh battery, etc.) is commissioned. The overall operation becomes simple if the newly commissioned UAV readily starts its job by streaming to the same resource at the fixed infrastructure. It can be easily configured to determine whether the consumer is up and watching by observing the responses to the CON requests. In case the exchange is initiated by the consumer then whenever a new UAV is commissioned, the consumer has to re-initiate the request again.

#### 5.1. Initial Negotiation

Initial negotiations for frame rate, video type, encoding details, etc., are performed by exchanging configuration scripts (cbor or json) over POST request. Exact format of the script is application dependent and is not part of this draft.

Fig. 3 illustrates the exemplary exchanges related to handshakes for connection initiation.

Note: All reliable transfers are in blocking mode. So, the producer MUST wait to send any further segment (critical/ on-critical) till the response is received for the critical segment. Please refer to Section 6 for suggested behavior in case a reliable transfer fails.

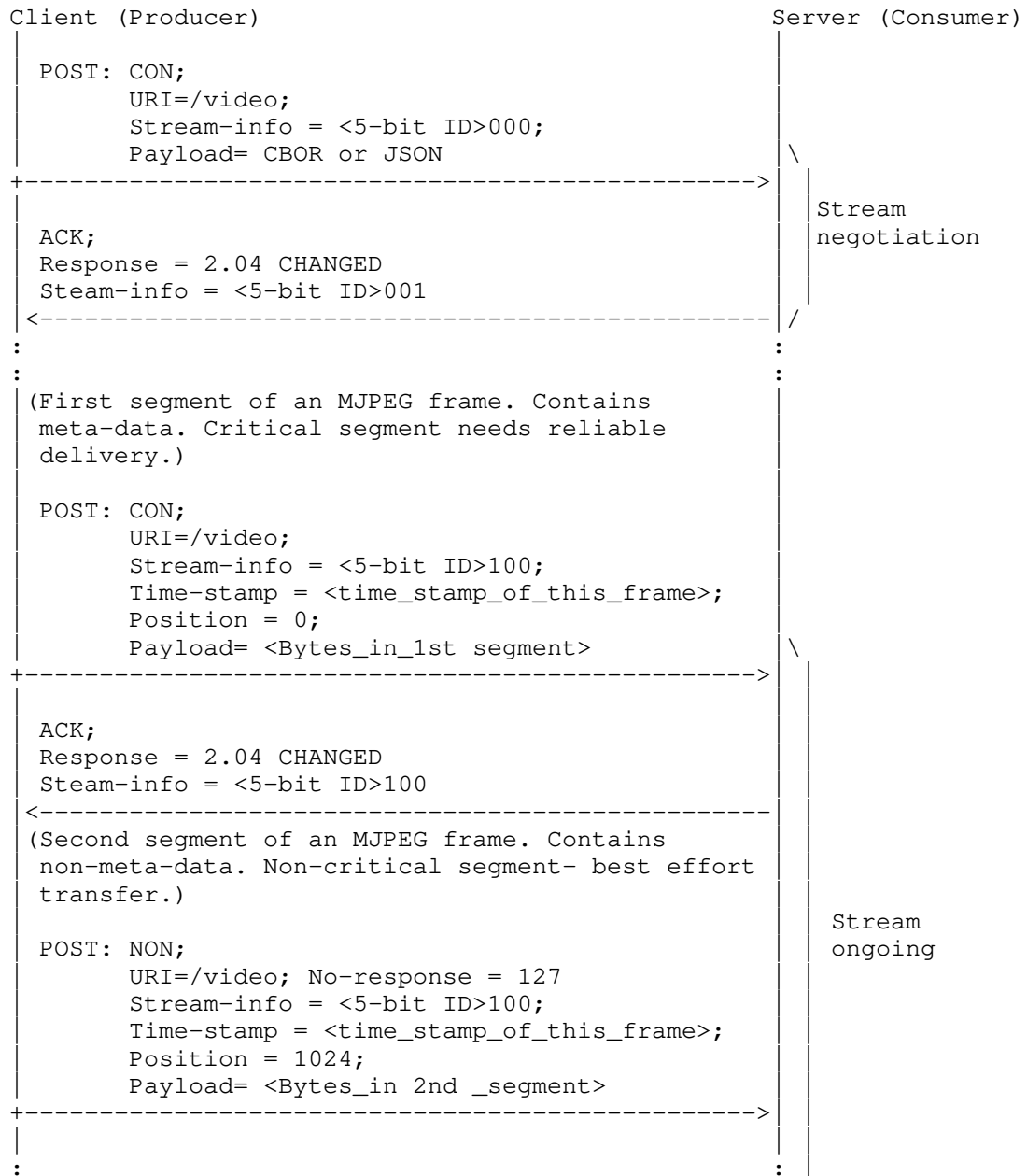


Figure 3: Example showing successful negotiation of streaming parameters followed by transmission of video information and control. It is assumed that the segment size negotiated as 1024 at the initiation. So, the position of the 2nd block is 1024. Note the use of No-response option with NON request for the non-critical segment.

## 5.2. Renegotiation

The renegotiation phase may occur when the "consumer" does not agree to parameters proposed by the producer and proposes a modified set. This may happen when the consumer application may need a less frame-rate than what is proposed by the producer. So, the "consumer" may request a lower frame-rate and thereby avoid unnecessary traffic in the network. The reduction may also be driven by the processing load on the producer which is anyway a constrained device. So, if a consumer requests more frame-rate than what is initially proposed by the producer, then the producer may insist on the lower frame-rate. Renegotiation may also occur if, during a stream, the producer senses a change in the end-to-end channel condition and proposes a new set of best possible parameters that can be served to the consumer.

Note that, that the consumer is never allowed to exceed the limits advertised by the producer.

Fig. 4 illustrates exemplary exchanges for re-negotiation.



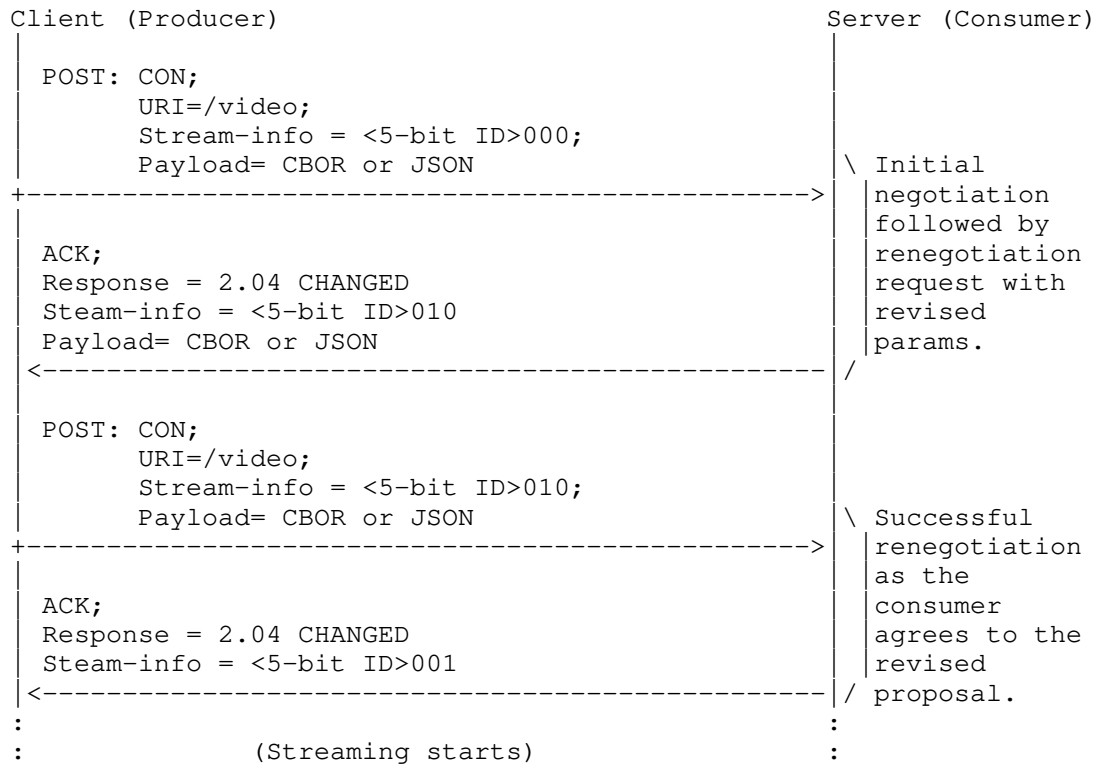


Figure 4: Example showing successful renegotiation of streaming parameters. Note the maneuvering of the Stream-info bit patterns.

Fig. 5 illustrates exemplary exchanges when a stream negotiation is unsuccessful. The accompanied script may provide hints to the reason for unsuccessful negotiations. A simple case of unsuccessful attempt may be observed if the resource on the "consumer" side is not ready. The exact formatting of the script is not in the scope of this draft.

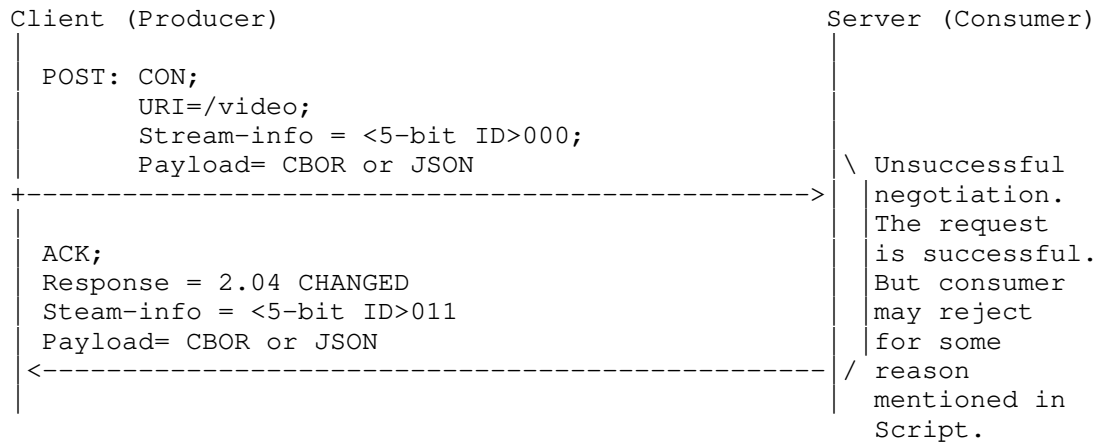


Figure 5: Example showing unsuccessful renegotiation despite successful response code against the initiation request.

## 6. Some Design Guidelines

### 6.1. Implicit Congestion Avoidance

The throughput and resource optimization for A-REaLiST depends largely on the best-effort delivery on UDP. Despite that the application designer can make A-REaLiST implicitly congestion aware and proactively avoid congestion. CoAP has a basic congestion avoidance mechanism which uses exponential back off to increase the timeout for retransmissions. However, that works only for CON messages.

The implicit congestion avoidance works like this: In case the producer fails to successfully transfer a critical segment of a frame within the MAX\_TRANSMIT\_SPAN as well as within MAX\_RETRANSMIT [RFC7252] attempts, the producer drops transmission of rest of the segments in that frame and waits for the next frame to be ready. The rationale is, since the critical segment is not delivered, the consumer will fail to reconstruct this frame anyway. So, there is no point in clogging the network with rest of the segments.

### 6.2. Considerations for Consumer-side Rendering

While the critical segments are delivered reliably in a sequential manner, non-critical are delivered with best-effort in an open-loop exchange. Also, the whole frame can be dropped to avoid congestion. Hence, the application at the "consumer" end-point (server) needs to

deal with issues like out-of-order delivery, frame/segment loss, asynchronous segment arrival.

The issues mentioned above have been discussed in literatures [Perkins]. So the basic approach should be: Buffer till a critical time to iron out the jittery, out-of-order arrival of the segments, play out from the appropriate buffer at a constant rate determined by the frame-rate of the video. There may be intelligent algorithms to play-out with high QoE despite non-arrival of non-critical segments within the play-out deadline. This draft provides the hooks to create such designs. Reference architecture of the play-out mechanism is provided in [Wi-UAV-Globecom]. The play-out architecture leverages on the design assumption about the 'less-constrained' nature of the consumer in terms of memory and processor.

### 6.3. Determining the segment size

Size of the information segment in a CoAP message should be limited by the least possible MTU for the end-to-end channel. This is to ensure that there is no undesired conversation state at the lower layers of the protocol stack due to uncontrolled fragmentation leading to undesired explosion of traffic in the network. For IPV6 network, the MTU can be determined using Path MTU Discovery (PMTUD) [RFC8201] which bestows the responsibility of determining the path MTU on the end-points itself.

The size of the segment should be guided by the recommendations as specified in Section 4.6 of [RFC7252].

### 7. IANA Considerations

The IANA is requested to assign numbers to the three options introduced in this draft for inclusion in the "CoAP Option Numbers" registry as shown below.

Number	Name	Reference
TBD	Stream-info	Section 4
TBD	Time-stamp	Section 4
TBD	Position	Section 4

## 8. Security Considerations

This draft presents no security considerations beyond those in Section 11 of the base CoAP specification [RFC7252].

## 9. References

### 9.1. Normative References

[RFC7252]

Shelby, Z., Hartke, K. and Bormann, C., "Constrained Application Protocol (CoAP)", RFC 7252, June, 2014.

[RFC7967]

Bhattacharyya, A., Bandyopadhyay, S., Pal, A., Bose, T., "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, August, 2016.

### 9.2. Informative References

[IOT-ISOC]

Rose, K., Eldridge, S., Chapin, L., "The Internet of Things: an overview", Internet Society, pp.1-50, October, 2015.

[RFC7452]

Tschafnig, H., Arkko, J., McPherson, D., "Architectural Considerations in Smart Object Networking", RFC 7452, March, 2015.

[Murphy]

Murphy, C., "Internet of Things: Are you underestimating video?", Available online:

<http://www.informationweek.com/bigdata/bigdataanalytics/internetofthingsareyouunderestimatingvideo/a/d-id/1269508>, June, 2014.

[Pereira]

Pereira, R., Pereira, E. G., "Video Streaming Considerations for Internet of Things", International Conference on Future Internet of Things and Cloud, pp. 48-52, August, 2014.

[RFC7959]

Bormann, C., Shelby, Z., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, August, 2016.

[GIoTS]

Dey, S., Bhattacharyya, A., Mukherjee, A., "Semantic data exchange between collaborative robots in fog environment: Can CoAP be a choice?", Global IOTS, pp. 1-6, June, 2017.

[RFC2616]

Fielding, R., Irvine, U.C., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June, 1999.

[RFC3550]

Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July, 2003.

[Wi-UAV-Globecom]

Bhattacharyya, A., Agrawal, S., Rath, H., Pal, A., "Improving Live-streaming Experience for Delay-sensitive IoT Applications : A RESTful Approach", accepted in Globecom (Wi-UAV workshop), Dec., 2018.

[Perkins]

Perkins, C., "RTP: Audio and Video for the Internet", Addison-Wesley, 2003.

[RFC8201]

McCann, J., et al., "Path MTU Discovery for IP version 6", RFC 8201, July, 2017.

Authors' Addresses

Abhijan Bhattacharyya  
Tata Consultancy Services Ltd.  
Kolkata, India

Email: abhijan.bhattacharyya@tcs.com

Suvrat Agrawal  
Tata Consultancy Services Ltd.  
Bangalore, India

Email: suvrat.a@tcs.com

Hemant Rath  
Tata Consultancy Services Ltd.  
Bhubaneswar, India

Email: hemant.rath@tcs.com

Arpan Pal  
Tata Consultancy Services Ltd.  
Kolkata, India

Email: arpan.pal@tcs.com

Balamurali Purushothaman  
Tata Consultancy Services Ltd.  
Bangalore, India

Email: balamurali.p@tcs.com





CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: July 6, 2019

H. Birkholz  
Fraunhofer SIT  
C. Bormann  
Universitaet Bremen TZI  
M. Pritikin  
Cisco  
R. Moskowitz  
Huawei  
January 02, 2019

Concise Identities  
draft-birkholz-core-coid-01

Abstract

There is an increased demand of trustworthy claim sets -- a set of system entity characteristics tied to an entity via signatures -- in order to provide information. Claim sets represented via CBOR Web Tokens (CWT) can compose a variety of evidence suitable for constrained-node networks and to support secure device automation. This document focuses on sets of identifiers and attributes that are tied to a system entity and are typically used to compose identities appropriate for Constrained RESTful Environment (CoRE) authentication needs.

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 July 6, 2019.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	4
2. Claims in a Concise Identity . . . . .	4
2.1. iss: CWT issuer . . . . .	5
2.2. sub: CWT subject . . . . .	5
2.3. aud: CWT audience . . . . .	5
2.4. exp: CWT expiration time . . . . .	5
2.5. nbf: CWT start of validity . . . . .	5
2.6. iat: CWT time of issue . . . . .	5
2.7. cti: CWT ID . . . . .	6
2.8. cnf: CWT proof-of-possession key claim . . . . .	6
3. Signature Envelope . . . . .	6
4. Processing Rules . . . . .	6
5. IANA Considerations . . . . .	6
6. Security Considerations . . . . .	6
7. References . . . . .	6
7.1. Normative References . . . . .	6
7.2. Informative References . . . . .	7
Appendix A. Common Terminology on Identity Documents . . . . .	8
A.1. Terms Specified in IEEE 802.1AR . . . . .	8
A.2. Terms Specified in RFC 4949 . . . . .	8
A.3. Terms specified in ISO/IEC 9594-8:2017 . . . . .	8
Appendix B. Concise Identities and Trust Relationships . . . . .	9
Appendix C. Concise Identity (CoID) CDDL Data Definition based on RFC 5280 . . . . .	10
Appendix D. Concise Secure Device Identifier (CoDeID) based on IEEE 802.1AR-2018 . . . . .	10
D.1. The Intended Use of DevIDs . . . . .	10
D.2. DevID Flavors . . . . .	11
D.3. Privacy . . . . .	12
D.4. Concise DevID CDDL data definition (sans COSE header) . . . . .	12
Appendix E. Attic . . . . .	14
E.1. Examples of claims taken from IEEE 802.1AR identifiers . . . . .	14
E.1.1. 7.2.1 version . . . . .	14
E.1.2. 7.2.2 serialNumber . . . . .	14
E.1.3. 7.2.3 signature . . . . .	14

E.1.4.	7.2.4 issuer Name . . . . .	15
E.1.5.	7.2.5 authoritykeyidentifier . . . . .	15
E.1.6.	7.2.7.1 notBefore . . . . .	15
E.1.7.	7.2.7.2 notAfter . . . . .	15
E.1.8.	7.2.8 subject . . . . .	15
E.1.9.	7.2.10 subjectPublicKeyInfo . . . . .	15
E.1.10.	7.2.11 signatureAlgorithm . . . . .	15
E.1.11.	7.2.12 signatureValue . . . . .	15
E.2.	Examples of claims taken from X.509 certificates . . . . .	16
E.2.1.	2.5.29.35 - Authority Key Identifier . . . . .	16
E.2.2.	2.5.29.14 - Subject Key Identifier . . . . .	16
E.2.3.	2.5.29.15 - Key Usage . . . . .	16
E.2.4.	2.5.29.37 - Extended key usage . . . . .	16
E.2.5.	1.3.6.1.5.5.7.1.1 - Authority Information Access . . . . .	16
E.2.6.	1.3.6.1.4.1.311.20.2 - Certificate Template Name Domain Controller (Microsoft) . . . . .	16
Appendix F.	Graveyard . . . . .	16
F.1.	7.2.9 subjectAltName . . . . .	17
F.2.	7.2.13 extensions . . . . .	17
F.3.	2.5.29.31 - CRL Distribution Points . . . . .	17
F.4.	2.5.29.17 - Subject Alternative Name . . . . .	17
F.5.	2.5.29.19 - Basic Constraints . . . . .	17
Acknowledgements	. . . . .	17
Authors' Addresses	. . . . .	17

## 1. Introduction

X.509 certificates [RFC5280] and Secure Device Identifier [IEEE-802.1AR] are ASN.1 encoded Identity Documents and intended to be tied to a system entity uniquely identified via these Identity Documents. An Identity Document - in general, a public-key certificate - can be conveyed to other system entities in order to prove or authenticate the identity of the owner of the Identity Document. Trust in the proof can be established by mutual trust of the provider and assessor of the identity in a third party verification (TVP) provided, for example, by a certificate authority (CA) or its subsidiaries (sub CA).

The evidence a certificate comprises is typically composed of a set of claims that is signed using secret keys issued by a (sub) CA. The core set of claims included in a certificate - its attributes - are well defined in the X.509v3 specifications and IEEE 802.1AR.

This document summarizes the core set of attributes and provides a corresponding list of claims using concise integer labels to be used in claim sets for CBOR Web Tokens (CWT) [RFC8392]. A resulting Concise Identity (CoID) is able to represent a signed set of claims that composes an Identity as defined in [RFC4949].

The objective of using CWT as a basis for the signed claim sets defined in this document is to gain more flexibility and at the same time more rigorously defined semantics for the signed claim sets. In addition, the benefits of using CBOR, COSE, and the corresponding CWT structure accrue, including more compact encoding and a simpler implementation in contrast to classical ASN.1 (DER/BER/PEM) structures and the X.509 complexity and uncertainty that has accreted since X.509 was released 29 years ago. One area where both the compactness and the definiteness are highly desirable is in Constrained-Node Networks [RFC7228], which may also make use of the Constrained Application Protocol (CoAP, [RFC7252]); however, the area of application of Concise Identities is not limited to constrained-node networks.

The present version of this document is a strawman that attempts to indicate the direction the work is intended to take. Not all inspirations this version takes from X.509 maybe need to be taken.

### 1.1. Terminology

This document uses terminology from [RFC8392] and therefore also [RFC7519], as well as from [RFC8152]. Specifically, we note:

Assertion:

Claim: A piece of information asserted about a subject. A claim is represented as a name/value pair consisting of a Claim Name and a Claim Value.

Claims are grouped into claims sets (represented here by a CWT), which need to be interpreted as a whole. Note that this usage is a bit different from idiomatic English usage, where a claim would stand on its own.

(Note that the current version of this draft is not very explicit about the relationship of identities and identifiers. To be done in next version.)

## 2. Claims in a Concise Identity

A Concise Identity (CoID) is a CBOR Web Token [RFC8392] with certain claims present. It can be signed in a number of ways, including a COSE\_Sign1 data object [RFC8152].

## 2.1. iss: CWT issuer

Optional: identifies the principal that is the claimant for the claims in the CoID ([RFC8392] Section 3.1.1, cf. Section 4.1.1 in [RFC7519]).

- o Note that this is a StringOrURI (if it contains a ":" it needs to be a URI)
- o For the "string" case (no ":"), there is no way to extract meaningful components from the string
- o Make it a URI if it needs to be structured (not for routine retrieval, unless specified so by an application)
- o If this URI looks like an HTTP or HTTPS URI then something retrievable by humans should exist there.
- o Alternatively, some arithmetic can be applied to the URI (extract origin, add /.well-known/...) to find relevant information.

## 2.2. sub: CWT subject

Optional: identifies the principal that is the subject for the claims in the CoID ([RFC8392] Section 3.1.2, cf. Section 4.1.2 in [RFC7519]).

## 2.3. aud: CWT audience

Optional: identifies the recipients that the CoID is intended for ([RFC8392] Section 3.1.4, cf. Section 4.1.4 in [RFC7519]).

## 2.4. exp: CWT expiration time

Optional: the time on or after which the CoID must no longer be accepted for processing ([RFC8392] Section 3.1.4, cf. Section 4.1.4 in [RFC7519]).

## 2.5. nbf: CWT start of validity

Optional: the time before which the CoID must not be accepted for processing ([RFC8392] Section 3.1.5, cf. Section 4.1.5 in [RFC7519]).

## 2.6. iat: CWT time of issue

Optional: the creation time of the CoID ([RFC8392] Section 3.1.6, cf. Section 4.1.6 in [RFC7519]).

## 2.7. cti: CWT ID

The "cti" (CWT ID) claim provides a unique identifier for the CoID ([RFC8392] Section 3.1.7, cf. "jti" in Section 4.1.7 in [RFC7519]).

CWT IDs are intended to be unique within an application, so they need to be either coordinated between issuers or based on sufficient randomness (e.g., 112 bits or more).

## 2.8. cnf: CWT proof-of-possession key claim

The "cnf" claim identifies the key that can be used by the subject for proof-of-possession and provides parameters to identify the CWT Confirmation Method ([I-D.ietf-ace-cwt-proof-of-possession] Section 3.1).

## 3. Signature Envelope

The signature envelope [TBD: need not actually be envelope, may be detached, too] carries additional information, e.g., the signature, as well as the identification of the signature algorithm employed (COSE: alg). Additional information may pertain to the signature (as opposed to the claims being signed), e.g., a key id (COSE: kid) may be given in the header of the signature.

## 4. Processing Rules

(TBD: This should contain some discussion of the processing rules that apply for CoIDs. Some of this will just be pointers to [I-D.ietf-oauth-jwt-bcp].)

## 5. IANA Considerations

This document makes no requests of IANA

## 6. Security Considerations

## 7. References

### 7.1. 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-05 (work in progress), November 2018.

- [I-D.ietf-oauth-jwt-bcp]  
Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", draft-ietf-oauth-jwt-bcp-04 (work in progress), November 2018.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

## 7.2. Informative References

- [IEEE-802.1AR]  
"IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity", IEEE standard, DOI 10.1109/ieeestd.2018.8423794, n.d..
- [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>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<https://www.rfc-editor.org/info/rfc5652>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

## Appendix A. Common Terminology on Identity Documents

To illustrate the purpose and intent of Identity Documents, typically, terms, such as certificates, certificate chains/paths and trust anchors, are used. To provide more context and for the convenience of the reader, three sources of definitions are highlighted in this section.

### A.1. Terms Specified in IEEE 802.1AR

1. a certificate is "a digitally signed object that binds information identifying an entity that possesses a secret private key to the corresponding public key."
2. a certificate chain is "an ordered list of intermediate certificates that links an end entity certificate ([...] a DevID certificate) to a trust anchor."
3. a trust anchor is "a Certificate Authority that is trusted and for which the trusting party holds information, usually in the form of a self-signed certificate issued by the trust anchor".

### A.2. Terms Specified in RFC 4949

1. a public-key certificate is "a digital certificate that binds a system entity's identifier to a public key value, and possibly to additional, secondary data items; i.e., a digitally signed data structure that attests to the ownership of a public key".
2. a certification path is "a linked sequence of one or more public-key certificates [...] that enables a certificate user to verify the signature on the last certificate in the path, and thus enables the user to obtain (from that last certificate) a certified public key, or certified attributes, of the system entity that is the subject of that last certificate".
3. a trust anchor is "a CA that is the subject of a trust anchor certificate or otherwise establishes a trust anchor key". Correspondingly, a trust anchor has a trust anchor certificate that "is a public-key certificate that is used to provide the first public key in a certification path".

### A.3. Terms specified in ISO/IEC 9594-8:2017

1. a public-key certificate is "the public key of an entity, together with some other information, rendered unforgeable by digital signature with the private key of the certification authority (CA) that issued it".



2. a certification path is "an ordered list of one or more public-key certificates, starting with a public-key certificate signed by the trust anchor, and ending with the end-entity public-key certificate to be validated. All intermediate public-key certificates, if any, are certification authority (CA) certificates in which the subject of the preceding public-key certificate is the issuer of the following public-key certificate".
3. a trust anchor is "an entity that is trusted by a relying party and used for validating public-key certificates".

#### Appendix B. Concise Identities and Trust Relationships

Following the terminology highlighted above, Concise Identities are signed CBOR Web Tokens that compose public-key Identity Documents based on asymmetric key pairs, potentially including additional assertions: claims that are secondary data items.

In the context of certification paths, the "last certificate" in the certification path is the Identity Document that resides on the system component, which presents its Identity Document to relying parties in order to be authenticated. The "first certificate" in the certification path resides on the trust anchor.

In order to be able to rely on the trust put into the Identity Document presented to relying parties, these have to put trust into two assumptions first:

- o the corresponding trust anchor (certificate) is trusted. In consequence, the consumer of the Identity Document requires a basis for decision whether to rely on the trust put in the trust anchor certificate, or not (e.g. via policies or a known certification paths).
- o the secret key included in the system component that is presenting its Identity Document is protected. In consequence, the secret key has to be stored in a shielded location. Type and quality of the protection or shielding or even its location are assertions that can be included as secondary data items in the Identity Document.

In summary, a path of trust relationships between a system component's Identity Document and a trusted authority's Identity Document is required to enable transitive trust in the system component that presents the Identity Document.

## Appendix C. Concise Identity (CoID) CDDL Data Definition based on RFC 5280

COSE MUST be used to sign this CoID template flavor.

"signatureAlgorithm" and "signature" are not part of the CoID map but of the COSE envelope.

```
CoID = { version: uint .range 1..3 ; (8)
        issuer: text, ; iss(1)
        subject: text / bytes, ; sub(2)
        notAfter: uint, ; exp(4)
        notBefore: uint ; nbf(5)
        serialNumber: uint, ; (7)
        subjectPublicKeyInfo: [ algorithm: COSE-Algorithm-Value,
                                subjectPublicKey: bytes,
                                ], ; (9)
        ? extensions: [ + [ extension-id: uint / registeredID,
                             extension-value: any,
                             ? criticality: bool,
                             ],
                        ], ; (0)
    }
```

```
COSE-Algorithm-Value = uint .size 0..2 / nint .size 0..2
registeredID = [ + uint ] ; OID
```

```
extensions = 0
issuer = 1
subject = 2
notAfter = 4
notBefore = 5
serialNumber = 7
version = 8
subjectPublicKeyInfo = 9
```

## Appendix D. Concise Secure Device Identifier (CoDeID) based on IEEE 802.1AR-2018

This section illustrates the context and background of Secure Device Identifiers.

### D.1. The Intended Use of DevIDs

IEEE 802.1AR Secure Device Identifier are a specific subset of X.509 Identity Documents that are intended to "authenticate a device's identity", where the corresponding Identity Document is "cryptographically bound to that device". In this context,

"cryptographically bound" means that the Identity Document is "constructed using cryptographic operations to combine a secret with other arbitrary data objects such that it may be proven that the result could only be created by an entity having knowledge of the secret."

While the intent of using X.509 Identity Documents as Device Identifiers starts to blur the line between authentication and authorization, the specification of IEEE 802.1AR Identity Documents provides a meaningful subset of assertions that can be used to identify one or more system components. The following CDDL data definition maps the semantics of an RFC 5280 Public Key Infrastructure Certificate Profile, which provides the basis for the Secure Device Identifier semantics. Both are mapped to a CWT representation.

#### D.2. DevID Flavors

In order to provide consistent semantics for the claims as defined below, understanding the distinction of IDevIDs (mandatory representation capabilities) and LDevIDs (recommended representation capabilities) is of the essence.

Both flavors of Secure Device Identifiers share most of their assertion semantics (claim sets).

IDevIDs are the initially Secure Device Identifiers that "are normally created during manufacturing or initial provisioning" and are "installed on the device by the manufacturer". IDevIDs are intended to be globally unique and to be stored in a way that protects it from modification (typically, a shielded location). It is important to note that a potential segregation of a manufacturer into separate supply chain/tree entities is not covered by the 802.1AR specification.

LDevIDs are the local significant Secure Device Identifiers that are intended to be "unique in the local administrative domain in which the device is used". In essence, LDevIDs "can be created at any time [after IDevID provisioning], in accordance with local policies". An "LDevID is bound to the device in a way that makes it infeasible for it to be forged or transferred to a device with a different IDevID without knowledge of the private key used to effect the cryptographic binding".

### D.3. Privacy

The exposition iof IDevID Identity Documents enables global unique identification of a system component. To mitigate the obvious privacy LDevIDs may also be used as the sole identifier (by disabling the IDevID) to assure the privacy of the user of a DevID and the equipment in which it is installed.

### D.4. Concise DevID CDDL data definition (sans COSE header)

COSE MUST be used to sign this DevID flavor, if represented via CoID.

"signature" and "signatureValue" are not part of the CoID map but of the COSE envelope.

"AlgorithmIdentifier" and corresponding "algorithm" and "parameters" should be part of the COSE envelope.

```

CoDeID = { version: 3, ;(8)
            serialNumber: uint,(7)
            issuer: text, ; iss(1)
            notAfter: uint, ; exp(4)
            notBefore: uint ; nbf(5)
            subject: text / URI, ; sub(2)
            subjectPublicKeyInfo: [ algorithm: COSE-Algorithm-Value,
                                   subjectPublicKey: bytes,
                                   ], ;(9)
            signatureAlgorithm: COSE-Algorithm-Value ; 802.1ar-2018 states
                                   ; this must be identical
                                   ; to cose sig-alg (rm?)
            authorityKeyIdentifier: bytes, ; all, non-critical,
            ? subjectKeyIdentifier: bytes, ; only intermediates, non-critical
            ? keyUsage : [ bitmask: bytes .size 1,
                          ? criticality: bool,
                          ]
            ? subjectAltName: text / iPAddress / registeredID,
            ? HardwareModuleName: [ hwType: registeredID,
                                   hwSerialNum: bytes,
                                   ],
            ? extensions: [ + [ extension-id: uint,
                                extension-value: any,
                                ? criticality: bool,
                                ],
                          ]
    }

```

COSE-Algorithm-Value = uint .size 0..2 / nint .size 0..2

iPAddress = bytes .size 4 / bytes .size 16

registeredID = [ + uint ] ; OID

extensions = 0

issuer = 1

subject = 2

notAfter = 4

notBefore = 5

serialNumber = 7

version = 8

subjectPublicKeyInfo = 9

signatureAlgorithm = 10

authorityKeyIdentifier = 11

subjectKeyIdentifier = 12

keyUsage = 13 ; could move to COSE header?

subjectAltName = 14

HardwareModuleName = 15

## Appendix E. Attic

Notes and previous content that will be pruned in next versions.

### E.1. Examples of claims taken from IEEE 802.1AR identifiers

This appendix briefly discusses common fields in a X.509 certificate or an IEEE 802.1AR Secure Device Identifier and relates them to claims in a CoID.

The original purpose of X.509 was only to sign the association between a name and a public key. In principle, if something else needs to be signed as well, CMS [RFC5652] is required. This principle has not been strictly upheld over time; this is demonstrated by the growth of various extensions to X.509 certificates that might or might not be interpreted to carry various additional claims.

This document details only the claim sets for CBOR Web Tokens that are necessary for authentication. The plausible integration or replacement of ASN.1 formats in enrollment protocols, [D]TLS handshakes and similar are not in scope of this document.

Subsections in this appendix are marked by the ASN.1 Object Identifier (OID) typically used for the X.509 item. [TODO: Make this true; there are still some section numbers.]

#### E.1.1. 7.2.1 version

The version field is typically not employed usefully in an X.509 certificate, except possibly in legacy applications that accept original (pre-v3) X.509 certificates.

Generally, the point of versioning is to deliberately inhibit interoperability (due to semantic meaning changes). CoIDs do not employ versioning. Where future work requires semantic changes, these will be expressed by making alternate kinds of claims.

#### E.1.2. 7.2.2 serialNumber

Covered by cti claim.

#### E.1.3. 7.2.3 signature

The signature, as well as the identification of the signature algorithm, are provided by the COSE container (e.g., COSE\_Sign1) used to sign the CoID's CWT.

## E.1.4. 7.2.4 issuer Name

Covered by iss claim.

## E.1.5. 7.2.5 authoritykeyidentifier

Covered by COSE kid in signature, if needed.

## E.1.6. 7.2.7.1 notBefore

Covered by nbf claim.

## E.1.7. 7.2.7.2 notAfter

Covered by exp claim.

For Secured Device identifiers, this claim is typically left out.

- o get a new one whenever you think you need it ("normal path")
- o nonced ocsf? might benefit from a more lightweight freshness verification of existing signed assertion - exploration required!
- o (first party only verifiable freshness may be cheaper than third-party verifiable?)

## E.1.8. 7.2.8 subject

Covered by sub claim.

Note that if claim sets need to be made about multiple subjects, the favored approach in CoID is to create multiple CoIDs, one each per subject.

## E.1.9. 7.2.10 subjectPublicKeyInfo

Covered by cnf claim.

## E.1.10. 7.2.11 signatureAlgorithm

In COSE\_Sign1 envelope.

## E.1.11. 7.2.12 signatureValue

In COSE\_Sign1 envelope.

## E.2. Examples of claims taken from X.509 certificates

Most claims in X.509 certificates take the form of certificate extensions. This section reviews a few common (and maybe not so common) certificate extensions and assesses their usefulness in signed claim sets.

### E.2.1. 2.5.29.35 - Authority Key Identifier

Used in certificate chaining. Can be mapped to COSE "kid" of the issuer.

### E.2.2. 2.5.29.14 - Subject Key Identifier

Used in certificate chaining. Can be mapped to COSE "kid" in the "cnf" (see Section 3.4 of [I-D.ietf-ace-cwt-proof-of-possession]).

### E.2.3. 2.5.29.15 - Key Usage

Usage information for a key claim that is included in the signed claims. Can be mapped to COSE "key\_ops" [TBD: Explain details].

### E.2.4. 2.5.29.37 - Extended key usage

Can include additional usage information such as 1.3.6.1.5.5.7.3.1 for TLS server certificates or 1.3.6.1.5.5.7.3.2 for TLS client certificates.

### E.2.5. 1.3.6.1.5.5.7.1.1 - Authority Information Access

More information about the signer. May include a pointer to signers higher up in the certificate chain (1.3.6.1.5.5.7.48.2), typically in the form of a URI to their certificate.

### E.2.6. 1.3.6.1.4.1.311.20.2 - Certificate Template Name Domain Controller (Microsoft)

This is an example for many ill-defined extensions that are on some arcs of the OID space somewhere.

E.g., the UCS-2 string (ASN.1 BMPString) "IPSECIntermediateOffline"

## Appendix F. Graveyard

Items and Content that was already discarded.



## F.1. 7.2.9 subjectAltName

(See "sub").

## F.2. 7.2.13 extensions

Extensions are handled by adding CWT claims to the CWT.

## F.3. 2.5.29.31 - CRL Distribution Points

Usually URIs of places where a CRL germane to the certificate can be obtained. Other forms of validating claim sets may be more appropriate than CRLs for the applications envisaged here.

(Might be replaced by a more general freshness verification approach later. For example one could define a generic "is this valid" request to an authority.)

## F.4. 2.5.29.17 - Subject Alternative Name

Additional names for the Subject.

These may be an "OtherName", i.e. a mystery blob "defined by" an ASN.1 OID such as 1.3.6.1.4.1.9.21.2.3, or one out of a few formats such as URIs (which may, then, turn out not to be really URIs). Naming subjects obviously is a major issue that needs attention.

## F.5. 2.5.29.19 - Basic Constraints

Can identify the key claim as that for a CA, and can limit the length of a certificate path. Empty in all the examples analyzed.

Any application space can define new fields / claims as appropriate and use them. There is no need for the underlying structure to define an additional extension method for this. Instead, they can use the registry as defined in Section 9.1 of [RFC8392].>

## Acknowledgements

## Authors' Addresses

Henk Birkholz  
Fraunhofer SIT  
Rheinstrasse 75  
Darmstadt 64295  
Germany

Email: [henk.birkholz@sit.fraunhofer.de](mailto:henk.birkholz@sit.fraunhofer.de)

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

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

Max Pritikin  
Cisco

Email: pritikin@cisco.com

Robert Moskowitz  
Huawei  
Oak Park, MI 48237

Email: rgm@labs.htt-consult.com

Network Working Group  
Internet-Draft  
Updates: 6690, 7252, 7641, 7959, 8132,  
8323 (if approved)  
Intended status: Standards Track  
Expires: April 27, 2019

C. Bormann  
Universitaet Bremen TZI  
October 24, 2018

Constrained Application Protocol (CoAP): Corrections and Clarifications  
draft-bormann-core-corr-clar-00

Abstract

RFC 7252 defines the Constrained Application Protocol (CoAP), along with a number of additional specifications, including RFC 7641, RFC 7959, RFC 8132, and RFC 8323. RFC 6690 defines the link format that is used in CoAP self-description documents.

Some parts of the specification may be unclear or even contain errors that may lead to misinterpretations that may impair interoperability between different implementations. The present document provides corrections, additions, and clarifications to the RFCs cited; this document thus updates these RFCs. In addition, other clarifications related to the use of CoAP in other specifications, including RFC 7390 and RFC 8075, are also provided.

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 27, 2019.

Copyright Notice

Copyright (c) 2018 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. Process . . . . .	3
1.2. Terminology . . . . .	3
2. RFC 7252 . . . . .	3
2.1. RFC7252-5.10.5: Max-Age . . . . .	4
3. IANA Considerations . . . . .	4
4. Security Considerations . . . . .	4
5. References . . . . .	4
5.1. Normative References . . . . .	4
5.2. Informative References . . . . .	5
Acknowledgements . . . . .	5
Author's Address . . . . .	6

## 1. Introduction

[RFC7252] defines the Constrained Application Protocol (CoAP), along with a number of additional specifications, including [RFC7641], [RFC7959], [RFC8132], and [RFC8323]. [RFC6690] defines the link format that is used in CoAP self-description documents.

During implementation and interoperability testing of these RFCs, and in their practical use, some ambiguities and common misinterpretations have been identified, as well as a few errors.

The present document summarizes identified issues and provides corrections needed for implementations of CoAP to interoperate, i.e., it constitutes an update to the RFCs referenced. This document also provides other clarifications related to common misinterpretations of the specification. References to CoAP should, therefore, also include this document.

In addition, some clarifications and corrections are also provided for documents that are related to CoAP, including RFC 7390 and RFC 8075.

## 1.1. Process

The present document is an Internet-Draft, which is not intended to be published as an RFC quickly. Instead, it will be maintained as a running document of the CoRE WG, probably for a number of years, until the need for new entries tails off and the document can finally be published as an RFC. (This paragraph to be rephrased when that happens.)

The status of this document as a running document of the WG implies a consensus process that is applied in making updates to it. The rest of this subsection provides more details about this consensus process. (This is the intended status; currently, the document is an individual submission only.)

(Consensus process TBD, but it will likely be based on an editor's version in a publicly accessible git repository, as well as periodic calls for consensus that lead to a new published Internet-Draft;.)

## 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.

When a section of this document makes formal corrections, additions or invalidations to text in a referenced RFC, this is clearly summarized. The text from the RFC that is being addressed is given and labeled "INCOMPLETE", "INCORRECT", or "INCORRECT AND INVALIDATED", followed by the correct text labeled "CORRECTED", where applicable. When text is added that does not simply correct text in previous specifications, it is given with the label "FORMAL ADDITION".

Where a resolution has not yet been agreed, the resolution is marked PENDING.

In this document, a reference to a section in RFC nnnn is written as RFC nnnn-<number>, where <number> is the section number.

## 2. RFC 7252

## 2.1. RFC7252-5.10.5: Max-Age

In the discussion of [I-D.ietf-core-too-many-reqs], a comment was made that it would be needed to define the point in time relative to which Max-Age is defined. A sender might reference it to the time it actually sends the message containing the option (and paragraph 3 of RFC7252-5.10.5 indeed requests that Max-Age be updated each time a message is retransmitted). The receiver of the message does not have reliable information about the time of sending, though. It may instead reference the Max-Age to the time of reception. This in effect extends the time of Max-Age by the latency of the packet. This extension was deemed acceptable for the purposes of [I-D.ietf-core-too-many-reqs], but may be suboptimal when Max-Age is about the lifetime of a response object.

INCOMPLETE:

The value is intended to be current at the time of transmission.

PENDING.

## 3. IANA Considerations

None yet.

(Individual clarifications may contain IANA considerations; these will then be referenced here.)

## 4. Security Considerations

This document provides a number of corrections and clarifications to existing RFCs, but it does not make any changes with regard to the security aspects of the protocol. As a consequence, the security considerations of the referenced RFCs apply without additions.

(To be changed when that is no longer true; probably the security considerations will then be on the individual clarifications.)

## 5. References

### 5.1. 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>>.

- [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>>.
- [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>>.
- [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>>.
- [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>>.

## 5.2. Informative References

- [I-D.ietf-core-too-many-reqs]  
Keranen, A., "Too Many Requests Response Code for the Constrained Application Protocol", draft-ietf-core-too-many-reqs-05 (work in progress), October 2018.

## Acknowledgements

The present document is modeled after RFC 4815 and the Internet-Drafts of the ROHC WG that led to it. Many thanks to the co-chairs of the ROHC WG and WG members that made this a worthwhile and successful experiment at the time.

Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

C. Bormann  
Universitaet Bremen TZI  
March 11, 2019

On Media-Types, Content-Types, and related terminology  
draft-bormann-core-media-content-type-format-00

Abstract

There is a lot of confusion about media-types, content-types, and related terminology.

This memo is an attempt at clearing it up.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	2
2. Media-Type . . . . .	2
3. Content-Type . . . . .	3
4. Content-Coding . . . . .	3
5. Content-Format . . . . .	4
6. Abbreviations . . . . .	4
7. Discussion . . . . .	4
8. IANA Considerations . . . . .	5
9. Security Considerations . . . . .	5
10. References . . . . .	5
10.1. Normative References . . . . .	5
10.2. Informative References . . . . .	5
Acknowledgements . . . . .	6
Author's Address . . . . .	6

## 1. Introduction

[RFC1590] introduced media types and their registration. That document took MIME types from [RFC1521] and gave them a new name. At that time, the term "media type" was often used just for the major type ("text", "audio"), and what we call a media-type now was the combination of a type and a subtype. This lives on in [RFC6838], which does not even have an ABNF production for media type:

```
type-name = reg-name
subtype-name = reg-name

reg-name = 1*127reg-name-chars
reg-name-chars = ALPHA / DIGIT / "!" /
                 "#" / "$" / "&" / "." /
                 "+" / "-" / "^" / "_"
```

## 2. Media-Type

However, the term "media type" is now generally used for a registered combination of a type-name and a subtype-name, as in

Media-Type = type-name "/" subtype

For the purposes of this memo, we define:

Media-Type: A combination of a type-name and a subtype-name registered in [IANA.media-types], conventionally identified by the two names separated by a slash.

### 3. Content-Type

Media types have parameters [RFC6838], some of which are mandatory. In HTTP and many other protocols, these are then used in a "Content-Type" header field. HTTP [RFC7231] uses:

```
Content-Type = media-type
media-type = type "/" subtype *( OWS ";" OWS parameter )
type       = token
subtype    = token
token      = 1*tchar
tchar      = "!" / "#" / "$" / "%" / "&" / "'" / "*"
           / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
           / DIGIT / ALPHA
OWS        = *( SP / HTAB )
```

We don't follow the inclusive use established by [RFC2616], parts of which became [RFC7231], to use the term media-type for a Media-Type with parameters; note that [RFC2616] was quite confused about this by claiming (Section 3.7):

Media-type values are registered with the Internet Assigned Number Authority (IANA [19]).

This clearly reverts to the understanding of Media-Type we use. We instead define as a separate term:

**Content-Type:** A Media-Type, optionally associated with parameters (separated from the media type name and from each other by a semicolon).

Removing the legacy HTAB characters now shunned in polite conversion, we define the conventional textual representation of a Content-Type as:

```
Content-Type = media-type *( *SP ";" *SP parameter )
```

### 4. Content-Coding

[RFC2616] also introduced the term Content-Coding, a registered name for an encoding transformation that has been or can be applied to a representation:

```
content-coding = token
```

Confusingly, in HTTP the Content-Coding is then given in a header field called "Content-Encoding"; we NEVER use this term (except when we are in error). Instead we define:

**Content-Coding:** a registered name for an encoding transformation that has been or can be applied to a representation.

Content-Codings are registered in the HTTP Content Coding Registry, a subregistry of [IANA.http-parameters]. We often use the "identity" Content-Coding, which is the identity transformation, and often fail to identify that Content-Coding by name, instead calling it "no Content-Coding".

## 5. Content-Format

CoAP [RFC7252] defines a Content-Format as the combination of a Content-Type and a Content-Coding, identified by a numeric identifier defined by the "CoAP Content-Formats" registry (a subregistry of [IANA.core-parameters]), but in more confusing words (it did not have the benefit of the present memo).

**Content-Format:** the combination of a Content-Type and a Content-Coding, identified by a numeric identifier defined by the "CoAP Content-Formats" registry.

Note that there is no conventional string representation of just the combination of a Content-Type and a Content-Coding; Content-Formats are always identified by their registered Content-Format numbers.

## 6. Abbreviations

Media-Types are sometime abbreviated as "mt", and Content-Types as "ct". We do not propose to use those abbreviations: Where the long form of the values can be used, the long form "Content-Type" can also be used to name them.

For historical reasons, both [RFC6690] and [RFC7252] use the abbreviation "ct" for Content-Format (think first and last character).

For Content-Coding, the abbreviation "cc" can be used.

## 7. Discussion

The ABNF given here is provisional and needs to be cleaned up: We need to unify the various forms of reg-name, token, etc. We need to define parameter. We also need to typographically differentiate foreign ABNF just shown for illustration from the normative ABNF of this memo.

We need to discuss case-insensitivity, which is usually rather insensitive.

## 8. IANA Considerations

While this memo talks a lot about IANA registries, it does not require any action from IANA.

## 9. Security Considerations

Confusion about terminology may, in the worst case, cause security problems. No other security considerations are known to be raised by the present memo.

## 10. References

### 10.1. Normative References

- [IANA.core-parameters]  
IANA, "Constrained RESTful Environments (CoRE) Parameters",  
<<http://www.iana.org/assignments/core-parameters>>.
- [IANA.http-parameters]  
IANA, "Hypertext Transfer Protocol (HTTP) Parameters",  
<<http://www.iana.org/assignments/http-parameters>>.
- [IANA.media-types]  
IANA, "Media Types",  
<<http://www.iana.org/assignments/media-types>>.

### 10.2. Informative References

- [RFC1521] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, DOI 10.17487/RFC1521, September 1993, <<https://www.rfc-editor.org/info/rfc1521>>.
- [RFC1590] Postel, J., "Media Type Registration Procedure", RFC 1590, DOI 10.17487/RFC1590, March 1994, <<https://www.rfc-editor.org/info/rfc1590>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<https://www.rfc-editor.org/info/rfc2616>>.

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

#### Acknowledgements

Matthias Kovatsch forced the author to make up his mind about this.  
Ari Keranen forced him to write it up, then.

#### Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 31, 2019

C. Bormann  
Universitaet Bremen TZI  
February 27, 2019

Additional Units for SenML  
draft-bormann-senml-more-units-00

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.

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 August 31, 2019.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	2
2. New Units . . . . .	2
3. Rationale . . . . .	3
4. Security Considerations . . . . .	3
5. IANA Considerations . . . . .	4
Acknowledgements . . . . .	4
7. Normative References . . . . .	4
Author's Address . . . . .	4

## 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.

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).

## 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
var	volt-ampere reactive (Reactive Power)	float	RFCthis
J/m	joule per meter (Energy per distance)	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 since that there is strong interest in using this unit specifically for the imaginary content of complex power, reactive power [IEEE-1459].

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. 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.

## 5. IANA Considerations

See Section 2.

## Acknowledgements

Ari Keranen pointed out the need for additional units in SenML.

## 7. 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.

[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>>.

## Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)

CoRE Working Group  
Internet-Draft  
Updates: 7390, 7641 (if approved)  
Intended status: Standards Track  
Expires: September 11, 2019

E. Dijk  
IoTconsultancy.nl  
C. Wang  
InterDigital  
M. Tiloca  
RISE AB  
March 10, 2019

Group Communication for the Constrained Application Protocol (CoAP)  
draft-dijk-core-groupcomm-bis-00

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 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 September 11, 2019.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Scope . . . . .	3
1.2. Terminology . . . . .	4
2. Use Cases . . . . .	4
2.1. Discovery . . . . .	4
2.1.1. Distributed Device Discovery . . . . .	4
2.1.2. Distributed Service Discovery . . . . .	5
2.1.3. Directory Discovery . . . . .	5
2.2. Operational . . . . .	5
2.2.1. Actuator Group Control . . . . .	6
2.2.2. Device Group Status Request . . . . .	6
2.2.3. Network-wide Query . . . . .	6
2.3. Software Update . . . . .	6
3. General Group Communication Operation . . . . .	7
3.1. Group Configuration . . . . .	7
3.1.1. Group Definition . . . . .	7
3.1.2. Group Naming (DNS) . . . . .	7
3.1.3. Group Creation and Membership . . . . .	8
3.1.4. Group Maintenance . . . . .	8
3.2. CoAP Usage . . . . .	9
3.2.1. Request/Response Model . . . . .	9
3.2.2. Port and URI Path Selection . . . . .	10
3.2.3. Proxy Operation . . . . .	11
3.2.4. Congestion Control . . . . .	11
3.2.5. Observing Resources . . . . .	11
3.2.6. Block-Wise Transfer . . . . .	11
3.3. Transport . . . . .	12
3.3.1. UDP/IPv6 Multicast Transport . . . . .	12
3.3.2. UDP/IPv4 Multicast Transport . . . . .	12
3.3.3. 6LoWPAN . . . . .	12
3.4. Interworking with Other Protocols . . . . .	12
3.4.1. MLD/MLDv2/IGMP . . . . .	12
3.4.2. RPL . . . . .	12
3.4.3. MPL . . . . .	12
4. Unsecured Group Communication . . . . .	12
5. Secured Group Communication using Group OSCORE . . . . .	13
5.1. Secure Group Maintenance . . . . .	14
6. Security Considerations . . . . .	15

6.1. CoAP NoSec Mode . . . . .	15
6.2. Group OSCORE . . . . .	15
6.3. 6LoWPAN . . . . .	16
6.4. Wi-Fi . . . . .	16
6.5. Monitoring . . . . .	17
7. IANA Considerations . . . . .	17
8. References . . . . .	17
8.1. Normative References . . . . .	17
8.2. Informative References . . . . .	18
Acknowledgments . . . . .	19
Authors' Addresses . . . . .	19

## 1. Introduction

This document specifies the use of the Constrained Application Protocol (CoAP) [RFC7252] for group communication [RFC7390]. CoAP is a RESTful communication protocol that is suited for usage in resource-constrained nodes, and in resource-constrained networks. This area of use is summarized as Constrained RESTful Environments (CoRE).

One-to-many group communication is achieved in CoAP by using UDP/IP multicast, as the underlying data transport to send multicast request messages. Multiple response messages to a single multicast request message are sent 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).

The most common use cases for group communication in resource-constrained networks are listed first, in Section 2. Both unsecured and secured CoAP group communication 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) [I-D.ietf-core-object-security]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [RFC8152] [RFC7049].

### 1.1. Scope

The guidelines and experimental protocol of [RFC7390] are updated by this document with the abovementioned security solution, and with other recent protocol developments around CoAP such as Observe [RFC7641] and Block-Wise Transfers [RFC7959].

For group communication, only solutions that use CoAP over UDP/multicast (both IPv6 and IPv4) are considered. Security solutions for group communication other than Group OSCORE are not in scope. General principles for secure group configuration are in scope, however a specific protocol for secure group configuration is not mandated because this is often application-specific.

## 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. Section 5 requires readers to be familiar with concepts and terminology from OSCORE [I-D.ietf-core-object-security] and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

## 2. 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 technical 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.

### 2.1. Discovery

Discovery of physical devices in a network, or discovery of information entities hosted on network devices, are operations that are particularly 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.

#### 2.1.1. Distributed Device Discovery

Device discovery is the discovery and identification of networked devices of a particular class, type, model, or brand. Group communication is used for distributed device discovery, where a central directory service is not used. Typically in distributed service 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 service is accessed through unicast, in which case group communication is not needed.

#### 2.1.2. 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 service not being used. Technically this is similar to the above use case of distributed device discovery (Section 2.1.1). For example, when using CoAP resource discovery (Section 7 of [RFC7252]) there is no technical distinction between doing distributed device discovery and distributed service discovery: both use the same CoAP query interface defined in Section 4 of [RFC6690].

#### 2.1.3. Directory Discovery

This use case is a specific sub-case of Distributed Service Discovery (Section 2.1.2), 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. One particular type of directory is the CoRE Resource Directory [I-D.ietf-core-resource-directory]; and there may be other types of directories that can be discovered using CoAP. 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 be looked up by other devices and/or applications.

### 2.2. Operational

Operational use cases describe those operations that occur most frequently in a networked system, during its operational lifetime and normal usage.

### 2.2.1. 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.

### 2.2.2. 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.

### 2.2.3. 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 (Section 2.1.2) 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.

## 2.3. Software Update

Multicast can be useful to efficiently distribute new software (firmware) 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 usually are 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 software blocks.



### 3. 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 3.1. Then the use of CoAP for group communication including support for protocol extensions (Block-Wise, Observe, PATCH method) follows in Section 3.2. How CoAP group messages are carried over various transport layers is the subject of Section 3.3. Finally, Section 3.4 covers the interworking of CoAP with other protocols at the layers below it.

#### 3.1. Group Configuration

##### 3.1.1. Group Definition

Following [RFC7390], a CoAP group is defined here 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. An endpoint may be a member of multiple 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 group.

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]).

##### 3.1.2. Group Naming (DNS)

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].

### 3.1.3. Group Creation and Membership

Group membership may be (factory-)preconfigured in devices or dynamically configured in a system on-site.

To create a group, a configuring entity defines an IP multicast address (or hostname) and a UDP port number for the group. Then it configures one or more devices as listeners to that IP multicast address, with a CoAP server listening on the specific port. These devices are the group members. The configuring entity can be a local application with preconfiguration, a user, a cloud service, or a local commissioning tool for example. 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.

The IETF does not define a mandatory, standardized protocol to accomplish these steps. 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 further defined in Section 5. [RFC7390] defines an experimental protocol for configuration of group membership for non-secured group communication, based on JSON-formatted configuration resources.

### 3.1.4. 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, 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.

When group communication is secured using Group OSCORE [I-D.ietf-core-oscore-groupcomm] (see Section 5), all CoAP endpoints participating to secure group communication MUST be members of a corresponding OSCORE group, and thus share a common set of cryptographic material. Additional maintenance operations are discussed in Section 5.1.

### 3.2. CoAP Usage

#### 3.2.1. Request/Response Model

Editor Note, TBD: this section is strongly based on Section 2.5 in [RFC7390]. In case a reference to this section is preferred, we can replace most of the following text in this section by a reference to it.

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 as detailed in Section 4.5 of [RFC7252].

A server MAY send back a unicast response to the CoAP group communication request – whether it does this or not is selected by the server application. 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.

TBD: the CoAP Option for No Server Response [RFC7967] can be used by the client to influence response suppression on the server side. Possibly we can include this information here; it specifically targets use for multicast use cases also.

The client can distinguish the origin of multiple server responses by the source IP address of the UDP message containing the CoAP response or any other available unique identifier (e.g., contained in the CoAP response payload). In case a client has sent multiple group requests with concurrent CoAP transactions ongoing, the responses are as usual matched to a request using the Token.

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

$\text{NON\_LIFETIME} + \text{MAX\_LATENCY} + \text{MAX\_SERVER\_RESPONSE\_DELAY}$

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 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.

### 3.2.2. 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 of 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 use cases where a request only can be successful for a subset of the group and errors are returned by those group members for which the request was unsuccessful.

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.

(TBD: consider if we should say that receiving node/server SHOULD NOT send a "ICMP Destination Unreachable - Port Unreachable" in response to such request.)

### 3.2.3. Proxy Operation

TBD: check if draft-ietf-core-multipart-ct-02 can solve the "multiple answers" case when a Proxy sends back multiple CoAP responses to a multicast request. Possibly a client may support this. Is there a way to signal multipart support by the client? Can the multipart parts signal the origin/IP address of their origin server?

### 3.2.4. Congestion Control

The measures to reduce network congestion risks are listed in Section 2.8 of [RFC7390], including both mandatory protocol elements as well as guidelines. This specification RECOMMENDS to apply the guidelines specified in that section.

### 3.2.5. 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.

Using the Observe Option in a CoAP multicast GET request is explicitly specified here as allowed; it is useful as a means to start observing a particular resource on all members of a (multicast) group at the same time. Group members that do not have this resource or do not allow the GET method on it will respond with the usual 4.04 Not Found or 4.05 Method Not Allowed, respectively.

A client sending a multicast GET with Observe MAY repeat this request using the same Token Option and Observe Option value, in order to ensure that enough (or all) group members have been reached with the request.

### 3.2.6. Block-Wise Transfer

Section 2.8 of [RFC7959] specifies how a server (group member), responding to a multicast request with a large resource, can use Block-Wise transfer to limit the size of the initial response.

TBD: investigate use of Block-Wise for PUT/POST/PATCH/iPATCH operations e.g. to be used for distributing software blocks over multicast. We can specify its use, or remark that this is undefined.

### 3.3. Transport

TBD: Mark [RFC8323] (TCP, TLS, WebSockets) as not applicable for this form of groupcomm, as well as CoAP-over-SMS.

#### 3.3.1. UDP/IPv6 Multicast Transport

TBD: include the "Exceptions" cases here of RFC 7390 Section 2.10. State that IPv6 multicast is prerequisite. Also mention the All-CoAP-nodes IPv6 addresses.

#### 3.3.2. UDP/IPv4 Multicast Transport

TBD: includes the "Exceptions" cases here of RFC 7390 2.10. State that IPv4 multicast is prerequisite. mention All-CoAP-nodes IPv4 addresses and the like

#### 3.3.3. 6LoWPAN

TBD: 6lowpan-specific considerations to go here. Specifically, a multicast request should preferably fit in one L2 frame to avoid the strong performance drop that comes with 6LoWPAN-fragmentation and reassembly. Also reference [RFC7346] for the realm-local scope.

### 3.4. Interworking with Other Protocols

#### 3.4.1. MLD/MLDv2/IGMP

TBD: see Section 4.2 of [RFC7390] and include the content here or refer to it.

#### 3.4.2. RPL

TBD: see Section 4.3 of [RFC7390] and include the content here or refer to it.

#### 3.4.3. MPL

TBD: see Section 4.4. [RFC7390] and include the content here or refer to it.

### 4. 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 6.1) must be well understood.

## 5. Secured Group Communication using Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] 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 Authenticated Encryption Algorithms with Additional Data (AEAD).

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 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 drives the join process for 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 3.1.1 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.

#### 5.1. Secure Group Maintenance

Additional key management operations on the OSCORE group are required, depending also on the security requirements of the application (see Section 6.2). 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].

## 6. Security Considerations

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

### 6.1. 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 non-critical applications (e.g., read-only temperature sensors where the client reading out the values 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.

### 6.2. Group OSCORE

Group OSCORE provides end-to-end application-level security. This has many desirable properties, including maintaining security properties 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.

A key management scheme for secure revocation and renewal of group keying material should be adopted in OSCORE groups. Also, 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]).

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.

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.

### 6.3. 6LoWPAN

Editor Note, TBD: identify if multi-fragment multicast requests have a negative effect on security and, if so, advice here on trying to avoid such requests. Also an attacker could use multi-fragment to occupy reassembly buffers of many routing 6LoWPAN nodes.

### 6.4. Wi-Fi

TBD: Wi-Fi specific security considerations; see also Section 5.3.1 of [RFC7390].

## 6.5. Monitoring

TBD: see Section 5.4 of [RFC7390].

## 7. IANA Considerations

This document has no actions for IANA.

## 8. References

### 8.1. Normative 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-oscore-groupcomm]  
Tiloca, M., Selander, G., Palombini, F., and J. Park,  
"Group OSCORE - Secure Group Communication for CoAP",  
draft-ietf-core-oscore-groupcomm-04 (work in progress),  
March 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>>.
- [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>>.
- [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>>.
- [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>>.

## 8.2. 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-01 (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-22 (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-19 (work in progress), January 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-01 (work in progress), January 2019.
- [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>>.
- [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>>.

#### Acknowledgments

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](mailto:esko.dijk@iotconsultancy.nl)

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

Email: [Chonggang.Wang@InterDigital.com](mailto:Chonggang.Wang@InterDigital.com)

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

Email: marco.tiloca@ri.se

Thing-to-Thing Research Group  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2019

K. Hartke  
Ericsson  
October 22, 2018

CoRE Applications  
draft-hartke-core-apps-08

Abstract

The application programmable interfaces of RESTful, hypermedia-driven Web applications consist of a number of reusable components such as Internet media types and link relation types. This document proposes "CoRE Applications", a convention for application designers to build the interfaces of their applications in a structured way, so that implementers can easily build interoperable clients and servers, and other designers can reuse the components in their own applications.

Note to Readers

This Internet-Draft should be discussed on the Thing-to-Thing Research Group (T2TRG) mailing list <t2trg@irtf.org> <<https://www.irtf.org/mailman/listinfo/t2trg>>.

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 25, 2019.

Copyright Notice

Copyright (c) 2018 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. CoRE Applications . . . . .	3
2.1. Communication Protocols . . . . .	4
2.1.1. URI Schemes . . . . .	4
2.2. Representation Formats . . . . .	5
2.2.1. Internet Media Types . . . . .	5
2.3. Links . . . . .	7
2.3.1. Link Relation Types . . . . .	8
2.3.2. Template Variable Names . . . . .	8
2.4. Forms . . . . .	8
2.4.1. Form Relation Types . . . . .	9
2.4.2. Form Field Names . . . . .	9
2.5. Well-Known Locations . . . . .	10
3. CoRE Application Descriptions . . . . .	10
3.1. Template . . . . .	11
4. URI Design Considerations . . . . .	12
5. Security Considerations . . . . .	14
6. IANA Considerations . . . . .	14
7. References . . . . .	14
7.1. Normative References . . . . .	14
7.2. Informative References . . . . .	15
Acknowledgements . . . . .	16
Author's Address . . . . .	17

## 1. Introduction

Representational State Transfer (REST) [16] is an architectural style for distributed hypermedia systems. Over the years, REST has gained popularity not only as an approach for large-scale information dissemination, but also as the basic principle for designing and building Internet-based applications in general.

In the coming years, the size and scope of the Internet is expected to increase greatly as physical-world objects become smart enough to communicate over the Internet -- a phenomenon known as the Internet of Things (IoT). As things learn to speak the languages of the net,



the idea of applying REST principles to the design of IoT application architectures suggests itself. To this end, the Constrained Application Protocol (CoAP) [23] was created, an application-layer protocol that enables RESTful applications in constrained-node networks [10], giving rise to a new setting for Internet-based applications: the Constrained RESTful Environment (CoRE).

To realize the full benefits and advantages of the REST architectural style, a set of constraints needs to be maintained when designing applications and their application programming interfaces (APIs). One of the fundamental principles of REST is that "REST APIs must be hypertext-driven" [17]. However, this principle is often ignored by application designers. Instead, APIs are specified out-of-band in terms of fixed URI patterns (e.g., in the API documentation or in a machine-readable format that facilitates code generation). Although this approach may appear easy for clients to use, the fixed resource names and data formats lead to a tight coupling between client and server implementations and make the system less flexible [5]. Violations of REST design principles like this result in APIs that may not be as scalable, extensible, and interoperable as promised by REST.

REST is intended for network-based applications that are long-lived and span multiple organizations [17]. Principled REST APIs require some design effort, since application designers do not only have to take current requirements into consideration, but also have to anticipate changes that may be required in the future -- years or even decades after the application has been deployed for the first time. The reward is long-term stability and evolvability, both of which are very desirable features in the Internet of Things.

To aid application designers in the design process, this document proposes "CoRE Applications", a convention for building the APIs of RESTful, hypermedia-driven Web applications. The goal is to help application designers avoid common mistakes by focusing almost all of the descriptive effort on defining the Internet media type(s) that are used for representing resources and driving application state.

A template for a "CoRE Application Description" provides a consistent format for the description of APIs so that implementers can easily build interoperable clients and servers, and other application designers can reuse the components in their own applications.

## 2. CoRE Applications

A CoRE Application API is a named set of reusable components. It describes a contract between a server hosting an instance of the

described application and clients that wish to interface with that instance.

The API is generally comprised of:

- o communication protocols, identified by URI schemes,
- o representation formats, identified by Internet media types,
- o link relation types,
- o form relation types,
- o template variables in templated links,
- o form field names in forms, and
- o well-known locations.

Together, these components provide the specific, in-band instructions to a client for interfacing with a given application.

## 2.1. Communication Protocols

The foundation of a hypermedia-driven REST API are the communication protocol(s) spoken between a client and a server. Although HTTP/1.1 [14] is by far the most common communication protocol for REST APIs, a REST API should typically not be dependent on any specific communication protocol.

### 2.1.1. URI Schemes

The usage of a particular protocol by a client is guided by URI schemes [7]. URI schemes specify the syntax and semantics of URI references [1] that the server includes in hypermedia controls such as links and forms.

A URI scheme refers to a family of protocols, typically distinguished by a version number. For example, the "http" URI scheme refers to the two members of the HTTP family of protocols: HTTP/1.1 [14] and HTTP/2 [8] (as well as some predecessors). The specific HTTP version used is negotiated between a client and a server in-band using the version indicator in the HTTP request-line or the TLS Application-Layer Protocol Negotiation (ALPN) extension [18].

IANA maintains a list of registered URI schemes at <http://www.iana.org/assignments/uri-schemes>.

## 2.2. Representation Formats

In RESTful applications, clients and servers exchange representations that capture the current or intended state of a resource and that are labeled with a media type. A representation is a sequence of bytes whose structure and semantics are specified by a representation format: a set of rules for encoding information.

Representation formats should generally allow clients with different goals, so they can do different things with the same data. The specification of a representation format "describes a problem space, not a prescribed relationship between client and server. Client and server must share an understanding of the representations they're passing back and forth, but they don't need to have the same idea of what the problem is that needs to be solved." [21]

Representation formats and their specifications frequently evolve over time. It is part of the responsibility of the designer of a new version to insure both forward and backward compatibility: new representations should work reasonably (with some fallback) with old processors and old representations should work reasonably with new processors [20].

Representation formats enable hypermedia-driven applications when they support the expression of hypermedia controls such as links (Section 2.3) and forms (Section 2.4).

### 2.2.1. Internet Media Types

One of the most important aspect of hypermedia-driven communications is the concept of Internet media types [2]. Media types are used to label representations so that it is known how the representation should be interpreted and how it is encoded. The centerpiece of a CoRE Application Description should be one or more media types.

**Note:** The terms media type and representation format are often used interchangeably. In this document, the term "media type" refers specifically to a string of characters such as "application/xml" that is used to label representations; the term "representation format" refers to the definition of the syntax and semantics of representations, such as XML 1.0 [12] or XML 1.1 [13].

A media type identifies a versioned series of representation formats (Section 2.2): a media type does not identify a particular version of a representation format; rather, the media type identifies the family, and includes provisions for version indicator(s) embedded in the representations themselves to determine more precisely the nature

of how the data is to be interpreted [20]. A new media type is only needed to designate a completely incompatible format [20].

Media types consist of a top-level type and a subtype, structured into trees [2]. Optionally, media types can have parameters. For example, the media type "text/plain; charset=utf-8" is a subtype for plain text under the "text" top-level type in the standards tree and has a parameter "charset" with the value "utf-8".

Media types can be further refined by

- o structured type name suffixes (e.g., "+xml" appended to the base subtype name; see Section 4.2.8 of RFC 6838 [2]),
- o a "profile" parameter (see Section 3.1 of RFC 6906 [24]),
- o subtype information embedded in the representations themselves (e.g., "xmlns" declarations in XML documents [11]),

or a similar annotation. An annotation directly in the media type is generally preferable, since subtype information embedded in representations can typically not be negotiated during content negotiation (e.g., using the CoAP Accept option).

In CoAP, media types are paired with a content coding [15] to indicate the "content format" [23] of a representation. Each content format is assigned a numeric identifier that can be used instead of the (more verbose) textual name of the media type in representation formats with size constraints. The flat number space loses the structural information that the textual names have, however.

The media type of a representation must be determined from in-band information (e.g., from the CoAP Content-Format option). Clients must not assume a structure from the application context or other out-of-band information.

IANA maintains a list of registered Internet media types at <http://www.iana.org/assignments/media-types>.

IANA maintains a list of registered structured suffixes at <http://www.iana.org/assignments/media-type-structured-suffix>.

IANA maintains a list of registered CoAP content formats at <http://www.iana.org/assignments/core-parameters>.

### 2.3. Links

As defined in RFC 8288 [6], a link is a typed connection between two resources. Additionally, a link is the primary means for a client to navigate from one resource to another.

A link is comprised of:

- o a link context,
- o a link relation type that identifies the semantics of the link (see Section 2.3.1),
- o a link target, identified by a URI, and
- o optionally, target attributes that further describe the link or the link target.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}, which has {target attributes}" [6]. For example, the resource <http://example.com/> could have a "terms-of-service" resource at <http://example.com/tos>, which has a representation with the media type "text/html".

There are two special kinds of links:

- o An embedding link is a link with an additional hint: when the link is processed, it should be substituted with the representation of the referenced resource rather than cause the client to navigate away from the current resource. Thus, traversing an embedding link adds to the current state rather than replacing it.

The most well known example for an embedding link is the HTML <img> element. When a Web browser processes this element, it automatically dereferences the "src" and renders the resulting image in place of the <img> element.

- o A templated link is a link where the client constructs the link target URI from provided in-band instructions. The specific rules for such instructions are described by the representation format. URI Templates [3] provide a generic way to construct URIs through variable expansion.

Templated links allow a client to construct resource URIs without being coupled to the resource structure at the server, provided that the client learns the template from a representation sent by the server and does not have the template hard-coded.

### 2.3.1. Link Relation Types

A link relation type identifies the semantics of a link [6]. For example, a link with the relation type "copyright" indicates that the resource identified by the target URI is a statement of the copyright terms applying to the link context.

Relation types are not to be confused with media types; they do not identify the format of the representation that results when the link is dereferenced [6]. Rather, they only describe how the link context is related to another resource [6].

IANA maintains a list of registered link relation types at <http://www.iana.org/assignments/link-relations>.

Applications that don't wish to register a link relation type can use an extension link relation type [6]: a URI that uniquely identifies the link relation type. For example, an application can use the string "http://example.com/foo" as link relation type without having to register it. Using a URI to identify an extension link relation type, rather than a simple string, reduces the probability of different link relation types using the same identifiers.

### 2.3.2. Template Variable Names

A templated link enables clients to construct the target URI of a link, for example, when the link refers to a space of resources rather than a single resource. The most prominent mechanisms for this are URI Templates [3] and the HTML <form> element with a submission method of GET.

To enable an automated client to construct an URI reference from a URI Template, the name of the variable in the template can be used to identify the semantics of the variable. For example, when retrieving the representation of a collection of temperature readings, a variable named "threshold" could indicate the variable for setting a threshold of the readings to retrieve.

Template variable names are scoped to link relation types, i.e., two variables with the same name can have different semantics if they appear in links with different link relation types.

## 2.4. Forms

A form is the primary means for a client to submit information to a server, typically in order to change resource state.

A form is comprised of:

- o a form context,
- o a form relation type that identifies the semantics of the form (see Section 2.4.1),
- o a request method (e.g., PUT, POST, DELETE),
- o a submission URI,
- o a description of a representation that the server expects as part of the form submission, and
- o optionally, target attributes that further describe the form or the form target.

A form can be viewed as an instruction of the form "To perform a {form relation type} operation on {form context}, make a {request method} request to {submission URI}, which has {target attributes}". For example, to "update" the resource <http://example.com/config>, a client would make a PUT request to <http://example.com/config>. (In many cases, the target of a form is the same resource as the context, but this is not required.)

The description of the expected representation can be a set of form fields (see Section 2.4.2) or simply a list of acceptable media types.

Note: A form with a submission method of GET is, strictly speaking, a templated link, since it provides a way to construct a URI and does not submit a representation to the server.

#### 2.4.1. Form Relation Types

A form relation type identifies the semantics of a form. For example, a form with the form relation type "create" indicates that a new item can be created within the form context by making a request to the resource identified by the target URI.

Similarly to extension link relation types, applications can use extension form relation types when they don't wish to register a form relation type.

#### 2.4.2. Form Field Names

Forms can have a detailed description of the representation expected by the server as part of form submission. This description typically consists of a set of form fields where each form field is comprised

of a field name, a field type, and optionally a number of attributes such as a default value, a validation rule or a human-readable label.

To enable an automated client to fill out a form, the field name can be used to identify the semantics of the form field. For example, when controlling a smart light bulb, the field name "brightness" could indicate the field for setting the desired brightness of the light bulb.

Field names are scoped to form relation types, i.e., two form fields with the same name can have different semantics if they appear in forms with different form relation types.

The type of a form field is a data type such as "an integer between 1 and 100" or "an RGB color". The type is orthogonal to the field name, i.e., the type should not be determined from the field name even though the client can identify the semantics of the field from the name. This separation makes it easy to change the set of acceptable values in the future.

## 2.5. Well-Known Locations

Some applications may require the discovery of information about a host, known as "site-wide metadata" in RFC 5785 [4]. For example, RFC 6415 [19] defines a metadata document format for describing a host; similarly, RFC 6690 [22] defines a link format for the discovery of resources hosted by a server.

Applications that need to define a resource for this kind of metadata can register new "well-known locations". RFC 5785 [4] defines the path prefix `"/.well-known/"` in "http" and "https" URIs for this purpose. RFC 7252 [23] extends this convention to "coap" and "coaps" URIs.

IANA maintains a list of registered well-known URIs at <http://www.iana.org/assignments/well-known-uris>.

## 3. CoRE Application Descriptions

As applications are implemented and deployed, it becomes important to describe them in some structured way. This section provides a simple template for CoRE Application Descriptions. A uniform structure allows implementers to easily determine the components that make up the interface of an application.

The template below lists all components of applications that both the client and the server implementation of the application need to understand in order to interoperate. Crucially, items not listed in



the template are not part of the contract between clients and servers -- they are implementation details. This includes in particular the URIs of resources (see Section 4).

CoRE Application Descriptions are intended to be published in human-readable format by designers of applications and by operators of deployed application instances. Application designers may publish an application description as a general specification of all application instances, so that implementers can create interoperable clients and servers. Operators of application instances may publish an application description as part of the API documentation of the service, which should also include instructions how the service can be located and which communication protocols and security modes are used.

### 3.1. Template

The fields of the template are as follows:

**Application name:**

Name of the application. The name is not used to negotiate capabilities; it is purely informational. A name may include a version number or, for example, refer to a living standard that is updated continuously.

**URI schemes:**

URI schemes identifying the communication protocols that need to be understood by clients and servers. This information is mostly relevant for deployed instances of the application rather than for the general specification of the application.

**Media types:**

Internet media types that identify the representation formats that need to be understood by clients and servers. An application description must comprise at least one media type. Additional media types may be required or optional.

**Link relation types:**

Link relation types that identify the semantics of links. An application description may comprise IANA-registered link relation types and extension link relation types. Both may be required or optional.

**Template variable names:**

For each link relation type, variable names that identify the semantics of variables in templated links with that link relation type. Whether a template variable is required or optional is indicated in-band inside the templated link.

**Form relation types:**

Form relation types that identify the semantics of forms and, for each form relation type, the submission method(s) to be used. An application description may comprise IANA-registered form relation types and extension form relation types. Both may be required or optional.

**Form field names:**

For each form relation type, form field names that identify the semantics of form fields in forms with that form relation type. Whether a form field is required or optional is indicated in-band inside the form.

**Well-known locations:**

Well-known locations in the resource identifier space of servers that clients can use to discover information given the DNS name or IP address of a server.

**Interoperability considerations:**

Any issues regarding the interoperable use of the components of the application should be given here.

**Security considerations:**

Security considerations for the security of the application must be specified here.

**Contact:**

Person (including contact information) to contact for further information.

**Author/Change controller:**

Person (including contact information) authorized to change this application description.

Each field should include full citations for all specifications necessary to understand the application components.

**4. URI Design Considerations**

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references 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 common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the

public REST API, prescribing a fixed format for this data. However, there are a number of problems with this practice [5], in particular if the application designer and the server owner are not the same entity.

In hypermedia-driven applications, URIs are therefore not included in the application interface. A CoRE Application Description must not mandate any particular form of URI substructure.

RFC 7320 [5] describes the problematic practice of fixed URI structures in detail and provides some acceptable alternatives.

Nevertheless, the design of the URI structure on a server is an essential part of implementing a RESTful application, even though it is not part of the application interface. The server implementer is responsible for binding the resources identified by the application designer to URIs.

A good RESTful URI is:

- o Short. Short URIs are easier to remember and cause less overhead in requests and representations.
- o Meaningful. A URI should describe the resource in a way that is meaningful and useful to humans.
- o Consistent. URIs should follow a consistent pattern to make it easy to reason about the application.
- o Bookmarkable. Cool URIs don't change [9]. However, in practice, application resource structures do change. That should cause URIs to change as well so they better reflect reality. Implementations should not depend on unchanging URIs.
- o Shareable. A URI should not be context sensitive, e.g., to the currently logged-in user. It should be possible to share a URI with third parties so they can access the same resource.
- o Extension-less. Some applications return different data for different extensions, e.g., for "contacts.xml" or "contacts.json". But different URIs imply different resources. RESTful URIs should identify a single resource. Different representations of the resource can be negotiated (e.g., using the CoAP Accept option).

## 5. Security Considerations

The security considerations of RFC 3986 [1], RFC 5785 [4], RFC 6570 [3], RFC 6838 [2], RFC 7320 [5], RFC 7595 [7], and RFC 8288 [6] are inherited.

All components of an application description are expected to contain clear security considerations. CoRE Application Descriptions should furthermore contain security considerations that need to be taken into account for the security of the overall application.

## 6. IANA Considerations

This document has no IANA actions.

## 7. References

### 7.1. Normative References

- [1] 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>>.
- [2] 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>>.
- [3] 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>>.
- [4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [5] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [6] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

- [7] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.

## 7.2. Informative References

- [8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [9] Berners-Lee, T., "Cool URIs don't change", 1998, <<http://www.w3.org/Provider/Style/URI.html>>.
- [10] 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>>.
- [11] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <<http://www.w3.org/TR/2009/REC-xml-names-20091208>>.
- [12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <<http://www.w3.org/TR/2006/REC-xml11-20060816>>.
- [14] 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>>.
- [15] 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>>.

- [16] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>>.
- [18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <<https://www.rfc-editor.org/info/rfc6415>>.
- [20] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.
- [21] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.
- [22] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [23] 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>>.
- [24] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.

#### Acknowledgements

Jan Algermissen, Mike Amundsen, Mike Kelly, Julian Reschke, and Erik Wilde provided valuable input on link and form relation types.

Thanks to Olaf Bergmann, Carsten Bormann, Stefanie Gerdes, Ari Keranen, Michael Koster, Matthias Kovatsch, Teemu Savolainen, and Bilhanan Silverajan for helpful comments and discussions that have shaped the document.

Some of the text in this document has been borrowed from [5], [6], [17], and [20]. All errors are my own.

This work was funded in part by Nokia.

Author's Address

Klaus Hartke  
Ericsson  
Torshamnsgatan 23  
Stockholm SE-16483  
Sweden

Email: klaus.hartke@ericsson.com

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

M. Koster  
SmartThings  
A. Keranen  
J. Jimenez  
Ericsson  
March 11, 2019

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)  
draft-ietf-core-coap-pubsub-08

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.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 13, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must



include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	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 . . . . .	6
4.1. DISCOVERY . . . . .	6
4.2. CREATE . . . . .	9
4.3. PUBLISH . . . . .	11
4.4. SUBSCRIBE . . . . .	14
4.5. UNSUBSCRIBE . . . . .	15
4.6. READ . . . . .	17
4.7. REMOVE . . . . .	18
5. CoAP Pub/sub Operation with Resource Directory . . . . .	20
6. Sleep-Wake Operation . . . . .	20
7. Simple Flow Control . . . . .	20
8. Security Considerations . . . . .	21
9. IANA Considerations . . . . .	22
9.1. Resource Type value 'core.ps' . . . . .	22
9.2. Resource Type value 'core.ps.discover' . . . . .	22
10. Acknowledgements . . . . .	22
11. References . . . . .	23
11.1. Normative References . . . . .	23
11.2. Informative References . . . . .	24
Authors' Addresses . . . . .	24

## 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

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 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.

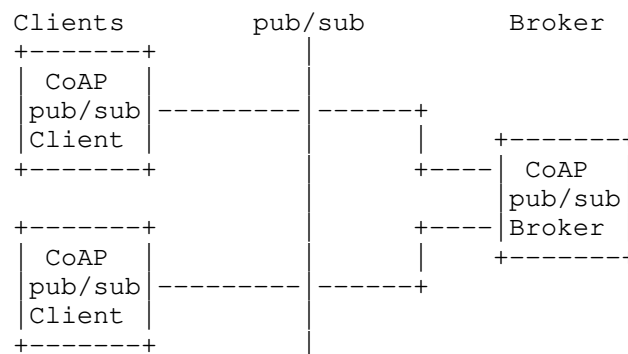


Figure 1: CoAP pub/sub Architecture

### 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 desirable to use pub/sub semantics for peer-to-peer communication, but it is not feasible or desirable to include a separate node on the network to serve as a Broker. In other use cases, it is desirable 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.

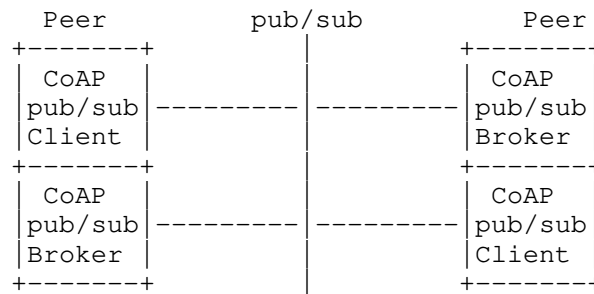


Figure 2: Brokerless pub/sub

## 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
<pre> ----- GET /.well-known/core?rt=core.ps ----&gt;&gt; -- Content-Format: application/link-format --- </pre>	<pre> &lt;&lt;--- 2.05 Content &lt;/ps/&gt;;rt=core.ps;rt=core.ps.discover;ct=40 --- </pre>

Figure 3: Example of DISCOVER pub/sub function

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

Figure 4: Example of DISCOVER topic

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

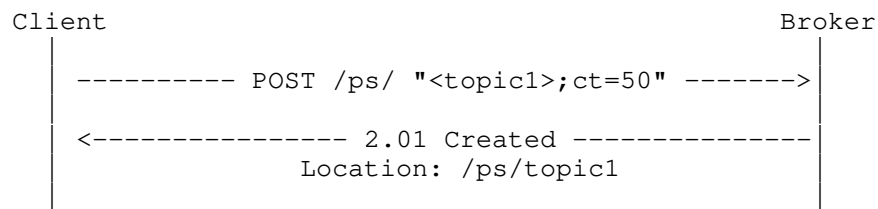


Figure 6: Example of CREATE topic

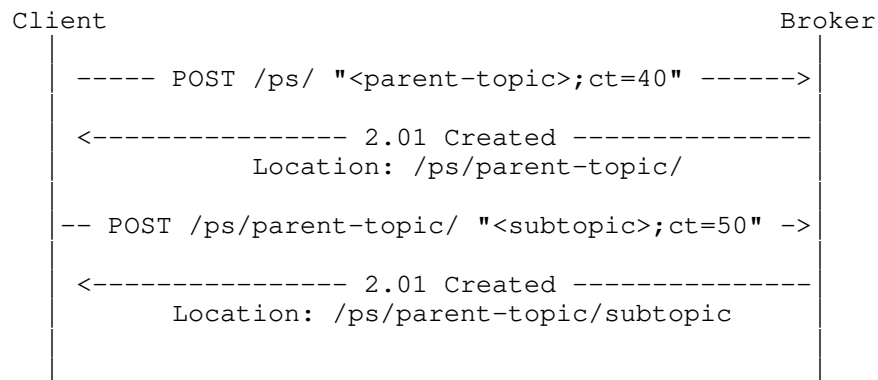


Figure 7: Example of CREATE of topic hierarchy

#### 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.

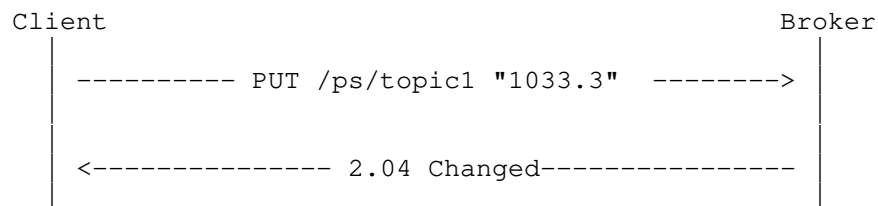


Figure 8: Example of PUBLISH

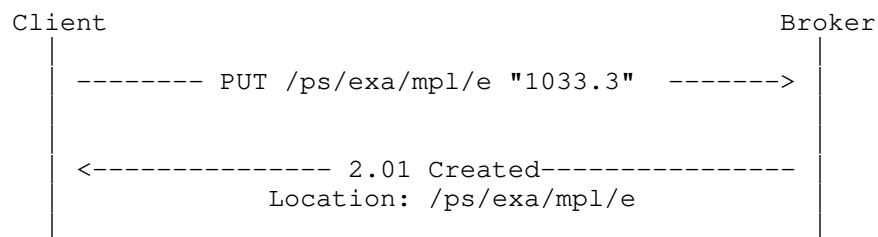


Figure 9: Example of CREATE on PUBLISH

#### 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.

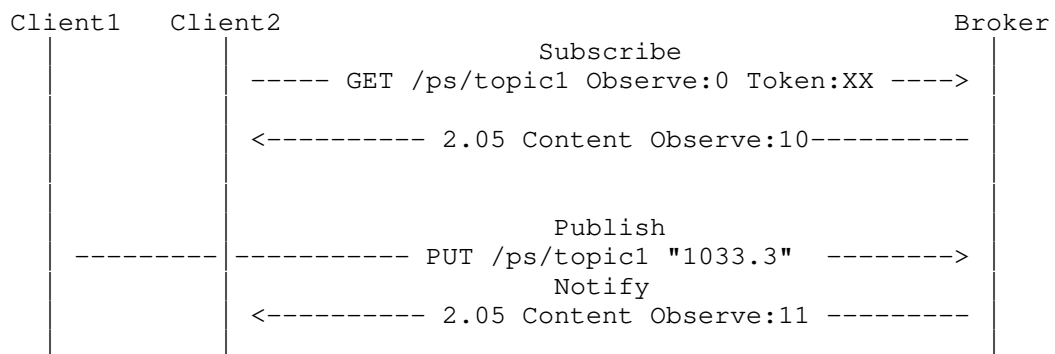


Figure 10: Example of SUBSCRIBE

#### 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.

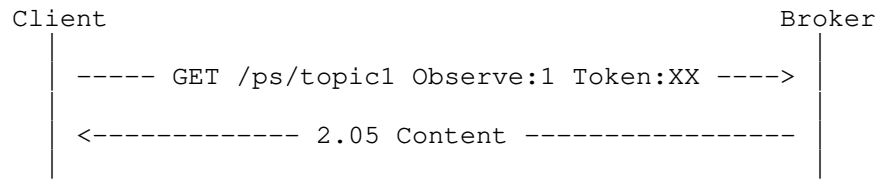


Figure 11: Example of UNSUBSCRIBE

#### 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.

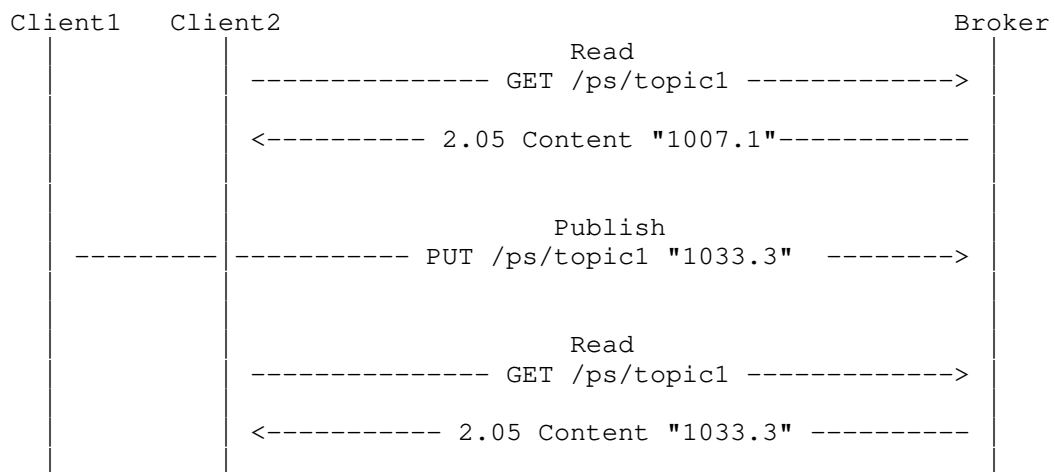


Figure 12: Example of READ

#### 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.

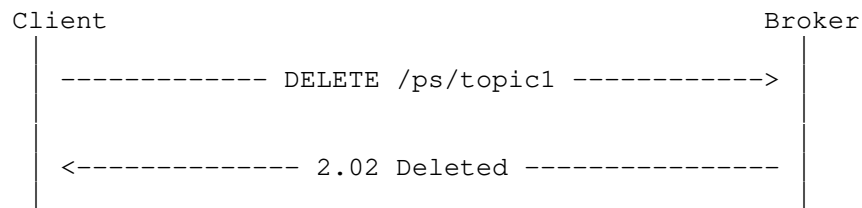


Figure 13: Example of REMOVE

## 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>>.
- [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>>.
- [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-20 (work in progress), March 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-03 (work in  
progress), June 2018.
- [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](mailto:Michael.Koster@smarththings.com)

Ari Keranen  
Ericsson

Email: [ari.keranen@ericsson.com](mailto:ari.keranen@ericsson.com)

Jaime Jimenez  
Ericsson

Email: [jaime.jimenez@ericsson.com](mailto:jaime.jimenez@ericsson.com)

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: May 16, 2019

M. Veillette, Ed.  
Trilliant Networks Inc.  
P. van der Stok, Ed.  
consultant  
A. Pelov  
Acklio  
A. Bierman  
YumaWorks  
November 12, 2018

CoAP Management Interface  
draft-ietf-core-comi-04

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. CoMI 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](mailto: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 May 16, 2019.



## Copyright Notice

Copyright (c) 2018 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 CoMI . . . . .	6
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 . . . . .	15
4.2.2. Using the 'd' Uri-Query option . . . . .	15
4.2.3. GET . . . . .	16
4.2.4. FETCH . . . . .	19
4.3. Data Editing . . . . .	20
4.3.1. Data Ordering . . . . .	20
4.3.2. POST . . . . .	20
4.3.3. PUT . . . . .	21
4.3.4. iPATCH . . . . .	22
4.3.5. DELETE . . . . .	23
4.4. Full datastore access . . . . .	23
4.4.1. Full datastore examples . . . . .	24
4.5. Event stream . . . . .	25
4.5.1. Notify Examples . . . . .	26
4.6. RPC statements . . . . .	27
4.6.1. RPC Example . . . . .	27
5. Access to MIB Data . . . . .	28
6. Use of Block . . . . .	28
7. Application Discovery . . . . .	29

7.1. YANG library . . . . .	29
7.2. Resource Discovery . . . . .	29
7.2.1. Datastore Resource Discovery . . . . .	30
7.2.2. Data node Resource Discovery . . . . .	30
7.2.3. Event stream Resource Discovery . . . . .	31
8. Error Handling . . . . .	31
9. Security Considerations . . . . .	34
10. IANA Considerations . . . . .	35
10.1. Resource Type (rt=) Link Target Attribute Values Registry . . . . .	35
10.2. CoAP Content-Formats Registry . . . . .	35
10.3. Media Types Registry . . . . .	36
11. Acknowledgements . . . . .	37
12. References . . . . .	38
12.1. Normative References . . . . .	38
12.2. Informative References . . . . .	39
Appendix A. ietf-comi YANG module . . . . .	40
Appendix B. ietf-comi .sid file . . . . .	46
Appendix C. YANG example specifications . . . . .	49
C.1. ietf-system . . . . .	49
C.2. server list . . . . .	51
C.3. interfaces . . . . .	52
C.4. Example-port . . . . .	52
C.5. IP-MIB . . . . .	53
Authors' Addresses . . . . .	55

## 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.

CoMI 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, CoMI 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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-yang-cbor]: 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.

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 schema node instance. Schema node 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.

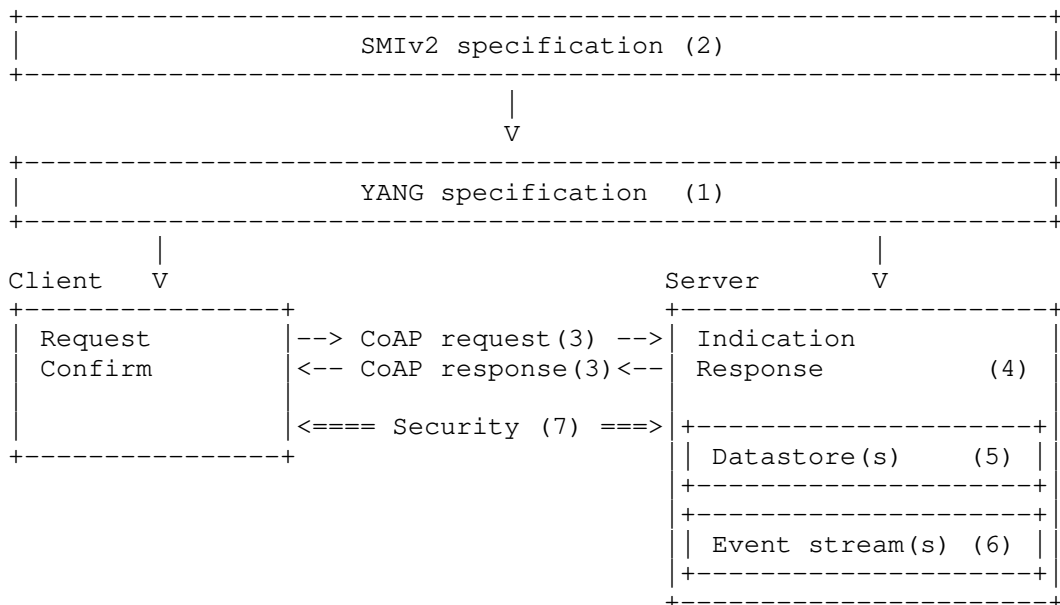


Figure 1: Abstract CoMI architecture

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: A named module 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] to secure CoAP communication.

#### 2.1. Major differences between RESTCONF and CoMI

CoMI 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, CoMI is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF. Some major differences are cited below:

- o CoMI 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 CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.
- o CoMI 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 used by CoAP.

- o CoMI does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.
- o CoMI does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.
- o CoMI does not support the "filter" query parameters to observe a specific set of notifications.
- o CoMI also differ in the handling of default values, only 'report-all' and 'trip' 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 are used instead of these strings. YANG Schema Item iDentifier (SID) is defined in [I-D.ietf-core-yang-cbor] section 2.1.

When used in a URI, SIDs are encoded in based64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5. 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.

```
SID in basae64 = 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]
```

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 of this CBOR map is composed of a key and a value. CBOR map keys are set to the SID assigned to the data nodes, CBOR map values are set to the instance value as 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 a single instance identifier and associated value. Instance identifiers are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1, 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-formats usage 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 on 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	CoMI
Defined in	"ietf-comi"

### 3. Example syntax

This section presents the notation used for the examples. The YANG modules that are used throughout this document are shown in Appendix C. The example modules are copied from existing modules and annotated with SIDs.

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. The path /c is recommended, but not compulsory (see Section 7).

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 resource 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:

Function	Recommended path	rt
Datastore	/c	core.c.ds
Data node	/c/SID	core.c.dn
Event steam	/s	core.c.ev

The path values are example values. 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 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. 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].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] may be used, as explained in more detail in Section 6.

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) parameter 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 parameter 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) parameter 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 values of a data node instance is sent with a CoAP GET message. An instance identifier is specified in the URI path prefixed with the example path /c.

FORMAT:

GET /c/instance-identifier

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

The returned payload contains the CBOR encoding of the specified data node instance value.

##### 4.2.3.1. GET Examples

Using for example the current-datetime leaf from Appendix C.1, a request is sent to retrieve the value of 'system-state/clock/current-datetime' specified in container system-state. 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 /  
 }  
}

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.

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 5.4.

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 data node 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. In this case, the format of each entry within the CBOR array of the FETCH response is identical to the format as a GET response.

##### FORMAT:

FETCH /c (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 instance-identifier, instance-value

##### 4.2.4.1. FETCH examples

This example uses the current-datetime leaf and the interface list from Appendix C.1. 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 SHOULD 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 resource is sent with a CoAP POST message. The URI specifies the data node to be instantiated at the exception of list instances. In this case, for compactness, the URI specifies the list for which an instance is created.

**FORMAT:**

```
POST /c/<instance identifier>
(Content-Format :application/yang-data+cbor)
CBOR map of SID, instance-value
```

2.01 Created

If the data node resource 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 Appendix C.1. Example is creating 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 /c/<instance identifier>
(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 Appendix C.1. Example is renewing an 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 iPATCH method [RFC8132]. A request is sent with a CoAP iPATCH message.

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 /c (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 : {                                  / server (SID 1756) /
      +3 : "tic.nrc.ca",                     / name (SID 1759) /
      +4 : true,                             / prefer (SID 1760) /
      +5 : {                                  / udp (SID 1761) /
        +1 : "132.246.11.231"                / address (SID 1762) /
      }
    }
  }
]
```

RES: 2.04 Changed

#### 4.3.5. DELETE

A data node resource is deleted with the DELETE method.

FORMAT:

Delete /c/<instance identifier>

2.02 Deleted

##### 4.3.5.1. DELETE example

This example uses the interface list from Appendix C.3. This example is deleting 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 /c

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

## FORMAT:

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

2.04 Changed

## FORMAT:

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

2.01 Created

## FORMAT:

DELETE /c

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 and the clock container from Appendix C.3. 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 : {
    / Clock (SID 1721) /
    +2: "2016-10-26T12:16:31Z", / current-datetime (SID 1723) /
    +1: "2014-10-05T09:00:00Z" / boot-datetime (SID 1722) /
  },
  1533 : [
    {
      / interface (SID 1533) /
      +4 : "eth0", / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880, / type (SID 1538), identity: /
      / ethernetCsmacd (SID 1880) /
      +2 : true / enabled (SID 1535) /
    }
  ]
}
```

#### 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 notification 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 contents is defined in [I-D.ietf-core-yang-cbor] section 4.2.1. For notification without any content, a null value is returned.

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 adjusted using an exclusion delay and limited by the maximum number of notifications supported, the



content of the instance is sent to all clients observing the modified stream.

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.

#### 4.5.1. Notify Examples

Suppose the server generates the event specified in Appendix C.4. By executing a GET on the /s resource the client receives the following response:

REQ: GET /s Observe(0) Token(0x93)

RES: 2.05 Content (Content-Format :application/yang-tree+cbor)  
Observe(12) Token(0x93)

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

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

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].

#### 4.6. 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 /c/<instance identifier>
      (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 specification of Appendix C.2. A server list is specified and the action "reset" (SID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
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. Access to MIB Data

Appendix C.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances.

REQ: GET example.com/c/Oz1

```
RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
  60021 : [
    {
      +1 : 1,          / ipNetToPhysicalIfIndex (SID 60022) /
      +2 : 1,          / ipNetToPhysicalNetAddressType (SID 60023) /
      +3 : h'0A000033', / ipNetToPhysicalNetAddress (SID 60024) /
      +4 : h'00000A01172D', / ipNetToPhysicalPhysAddress (SID 60025) /
      +5 : 2333943,     / ipNetToPhysicalLastUpdated (SID 60026) /
      +6 : 4,           / ipNetToPhysicalType (SID 60027) /
      +7 : 1,           / ipNetToPhysicalState (SID 60028) /
      +8 : 1            / ipNetToPhysicalRowStatus (SID 60029) /
    },
    {
      +1 : 1,          / ipNetToPhysicalIfIndex (SID 60022) /
      +2 : 1,          / ipNetToPhysicalNetAddressType (SID 60023) /
      +3 : h'09020304', / ipNetToPhysicalNetAddress (SID 60024) /
      +4 : h'00000A36200A', / ipNetToPhysicalPhysAddress (SID 60025) /
      +5 : 2329836,     / ipNetToPhysicalLastUpdated (SID 60026) /
      +6 : 3,           / ipNetToPhysicalType (SID 60027) /
      +7 : 6,           / ipNetToPhysicalState (SID 60028) /
      +8 : 1            / ipNetToPhysicalRowStatus (SID 60029) /
    }
  ]
}
```

## 6. Use of Block

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 the complete data field.

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.

## 7. Application Discovery

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

### 7.1. YANG library

The YANG library data model [I-D.veillette-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.

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.

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

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

### 7.2. Resource Discovery

Even if the YANG library provides all the information needed for application discovery, the implementation of Resource discovery as defined by [RFC6690] can be desirable for a seamless integration with other CoAP interfaces and services.

### 7.2.1. 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. 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
```

### 7.2.2. Data node Resource Discovery

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"
```

The list of data nodes may become prohibitively long. Implementations MAY return a subset of this list or can rely solely on the YANG library.

### 7.2.3. 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"
```

## 8. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message 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 don't match those specified in section Section 2.4.

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 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.
  - \* 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 don't follow the rules set by or when the value is incompatible with the YANG Built-In type. (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 fail 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 an 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) /
  }
}
```

## 9. 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]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However, some adaptations may still be required, to cater for CoMI's specific requirements.

## 10. IANA Considerations

### 10.1. 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.

### 10.2. 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	Encoding ID	Reference
application/yang-data+cbor	XXX	RFC XXXX
application/yang-identifiers+cbor	XXX	RFC XXXX
application/yang-instances+cbor	XXX	RFC XXXX

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

### 10.3. 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.

## 11. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes.

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.

## 12. References

### 12.1. Normative References

- [I-D.ietf-core-sid]  
Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-04 (work in progress), June 2018.
- [I-D.ietf-core-yang-cbor]  
Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-07 (work in progress), September 2018.
- [I-D.veillette-core-yang-library]  
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-yang-library-03 (work in progress), September 2018.
- [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>>.

## 12.2. Informative References

- [RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <<https://www.rfc-editor.org/info/rfc4293>>.
- [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>>.

#### Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2018-09-26.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 only present 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.";

revision 2018-09-26 {
  description
    "Use of YANG data template for the error payload.
    Definition of the unified datastore.";
  reference
    "[I-D.ietf-core-comi] CoAP Management Interface";
}
revision 2017-07-01 {
  description
    "Initial revision.";
  reference
    "[I-D.ietf-core-comi] CoAP Management Interface";
}

typedef sid {
  type uint64;
  description
    "YANG Schema Item iDentifier";
  reference
    "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

identity unified {
  base ds:datastore;
  description
    "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": "2018-09-26",  
  "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",
    "identifier": "/ietf-comi:error/error-tag",
    "sid": 1028
  }
]
}

```

## Appendix C. YANG example specifications

This appendix shows five YANG example specifications taken over from as many existing YANG modules. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign.

### C.1. ietf-system

Excerpt of the YANG module ietf-system [RFC7317].

```

module ietf-system {
    container system {
        container clock {
            choice timezone {
                case timezone-name {
                    leaf timezone-name {
                        type timezone-name;
                    }
                }
                case timezone-utc-offset {
                    leaf timezone-utc-offset {

```



```
        type int16 {
        }
    }
}
container ntp { // SID 1754
    leaf enabled { // SID 1755
        type boolean;
        default true;
    }
    list server { // SID 1756
        key name;
        leaf name { // SID 1759
            type string;
        }
        choice transport {
            case udp {
                container udp { // SID 1761
                    leaf address { // SID 1762
                        type inet:host;
                    }
                    leaf port { // SID 1763
                        type inet:port-number;
                    }
                }
            }
        }
    }
    leaf association-type { // SID 1757
        type enumeration {
            enum server {
            }
            enum peer {
            }
            enum pool {
            }
        }
    }
    leaf iburst { // SID 1758
        type boolean;
    }
    leaf prefer { // SID 1760
        type boolean;
        default false;
    }
}
container system-state { // SID 1720
```

```
        container clock {                                // SID 1721
            leaf current-datetime {                       // SID 1723
                type yang:date-and-time;
            }
            leaf boot-datetime {                          // SID 1722
                type yang:date-and-time;
            }
        }
    }
}
```

## C.2. server list

Taken over from [RFC7950] section 7.15.3.

```
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;
                }
            }
        }
    }
}
```

### C.3. interfaces

Excerpt of the YANG module ietf-interfaces [RFC7223].

```
module ietf-interfaces {                               // SID 1500
  container interfaces {                               // SID 1505
    list interface {                                   // SID 1533
      key "name";
      leaf name {                                     // SID 1537
        type string;
      }
      leaf description {                             // SID 1534
        type string;
      }
      leaf type {                                     // SID 1538
        type identityref {
          base interface-type;
        }
        mandatory true;
      }

      leaf enabled {                                  // SID 1535
        type boolean;
        default "true";
      }

      leaf link-up-down-trap-enable { // SID 1536
        if-feature if-mib;
        type enumeration {
          enum enabled {
            value 1;
          }
          enum disabled {
            value 2;
          }
        }
      }
    }
  }
}
```

### C.4. Example-port

Notification example defined within this document.

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

#### C.5. IP-MIB

The YANG translation of the SMI specifying the IP-MIB [RFC4293], extended with example SID numbers, yields:

```
module IP-MIB {  
    import IF-MIB {  
        prefix if-mib;  
    }  
    import INET-ADDRESS-MIB {  
        prefix inet-address;  
    }  
    import SNMPv2-TC {  
        prefix smiv2;  
    }  
    import ietf-inet-types {  
        prefix inet;  
    }  
    import yang-smi {  
        prefix smi;  
    }  
    import ietf-yang-types {  
        prefix yang;  
    }  
  
    container ip {  
        list ipNetToPhysicalEntry { // SID 60020  
            key "ipNetToPhysicalIfIndex  
                ipNetToPhysicalNetAddressType  
                ipNetToPhysicalNetAddress";  
            leaf ipNetToPhysicalIfIndex { // SID 60022
```

```
        type if-mib:InterfaceIndex;
    }
    leaf ipNetToPhysicalNetAddressType { // SID 60023
        type inet-address:InetAddressType;
    }
    leaf ipNetToPhysicalNetAddress { // SID 60024
        type inet-address:InetAddress;
    }
    leaf ipNetToPhysicalPhysAddress { // SID 60025
        type yang:phys-address {
            length "0..65535";
        }
    }
    leaf ipNetToPhysicalLastUpdated { // SID 60026
        type yang:timestamp;
    }
    leaf ipNetToPhysicalType { // SID 60027
        type enumeration {
            enum "other" {
                value 1;
            }
            enum "invalid" {
                value 2;
            }
            enum "dynamic" {
                value 3;
            }
            enum "static" {
                value 4;
            }
            enum "local" {
                value 5;
            }
        }
    }
    leaf ipNetToPhysicalState { // SID 60028
        type enumeration {
            enum "reachable" {
                value 1;
            }
            enum "stale" {
                value 2;
            }
            enum "delay" {
                value 3;
            }
            enum "probe" {
                value 4;
            }
        }
    }
}
```

```
    }
    enum "invalid" {
        value 5;
    }
    enum "unknown" {
        value 6;
    }
    enum "incomplete" {
        value 7;
    }
}
}
leaf ipNetToPhysicalRowStatus {          // SID 60029
    type smiv2:RowStatus;
} // list ipNetToPhysicalEntry
} // container ip
} // module IP-MIB
```

#### Authors' Addresses

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

Email: [michel.veillette@trilliant.com](mailto:michel.veillette@trilliant.com)

Peter van der Stok (editor)  
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)  
Email: [consultancy@vanderstok.org](mailto:consultancy@vanderstok.org)  
URI: [www.vanderstok.org](http://www.vanderstok.org)

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

Email: [a@ackl.io](mailto:a@ackl.io)

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

Email: [andy@yumaworks.com](mailto:andy@yumaworks.com)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2019

J. Arkko  
Ericsson  
C. Jennings  
Cisco  
Z. Shelby  
ARM  
October 22, 2018

Uniform Resource Names for Device Identifiers  
draft-ietf-core-dev-urn-03

Abstract

This memo describes a new Uniform Resource Name (URN) namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories. A URN-based representation can be easily passed along in any application that needs the information.

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 25, 2019.

Copyright Notice

Copyright (c) 2018 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. Requirements language . . . . .	3
3. DEV URN Definition . . . . .	4
3.1. Purpose . . . . .	4
3.2. Syntax . . . . .	5
3.3. Assignment . . . . .	6
3.4. Security and Privacy . . . . .	6
3.5. Interoperability . . . . .	6
3.6. Resolution . . . . .	6
3.7. Documentation . . . . .	6
3.8. Additional Information . . . . .	7
3.9. Revision Information . . . . .	7
4. DEV URN Subtypes . . . . .	7
4.1. MAC Addresses . . . . .	7
4.2. 1-Wire Device Identifiers . . . . .	7
4.3. Organization-Defined Identifiers . . . . .	8
4.4. Organization Serial Numbers . . . . .	8
4.5. Organization Product and Serial Numbers . . . . .	8
5. Examples . . . . .	8
6. Security Considerations . . . . .	9
7. IANA Considerations . . . . .	10
8. References . . . . .	10
8.1. Normative References . . . . .	10
8.2. Informative References . . . . .	11
Appendix A. Changes from Previous Version . . . . .	13
Appendix B. Acknowledgments . . . . .	14
Authors' Addresses . . . . .	14

## 1. Introduction

This memo describes a new Uniform Resource Name (URN) [RFC8141] [RFC3406] namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories [RFC7252], [RFC8428]. A URN-based representation can be easily passed along in any application that needs the information, as it fits in protocols mechanisms that are designed to carry URNs [RFC2616], [RFC3261], [RFC7252]. Finally, URNs can also be easily carried and stored in formats such as XML [W3C.REC-xml-19980210] or JSON [RFC8428] [RFC4627]. Using URNs in these formats is often preferable as they are universally recognized, self-describing, and

therefore avoid the need for agreeing to interpret an octet string as a specific form of a MAC address, for instance.

This memo defines identity URN types for situations where no such convenient type already exist. For instance, [RFC6920] defines cryptographic identifiers, [RFC7254] defines International Mobile station Equipment Identity (IMEI) identifiers for use with 3GPP cellular systems, and [I-D.atarius-dispatch-meid-urn] defines Mobile Equipment Identity (MEID) identifiers for use with 3GPP2 cellular systems. Those URN types should be employed when such identities are transported; this memo does not redefine these identifiers in any way.

Universally Unique Identifier (UUID) URNs [RFC4122] are another alternative way for representing device identifiers, and already support MAC addresses as one of type of an identifier. However, UUIDs can be inconvenient in environments where it is important that the identifiers are as simple as possible and where additional requirements on stable storage, real-time clocks, and identifier length can be prohibitive. UUID-based identifiers are recommended for all general purpose uses when MAC addresses are available as identifiers. The device URN defined in this memo is recommended for constrained environments.

Future device identifier types can extend the device device URN type defined here, or define their own URNs.

Note that long-term stable unique identifiers are problematic for privacy reasons and should be used with care or avoided as described in [RFC7721].

The rest of this memo is organized as follows. Section 3 defines the "DEV" URN type, and Section 4 defines subtypes for IEEE MAC-48, EUI-48 and EUI-64 addresses and 1-wire device identifiers. Section 5 gives examples. Section 6 discusses the security considerations of the new URN type. Finally, Section 7 specifies the IANA registration for the new URN type and sets requirements for subtype allocations within this type.

## 2. Requirements language

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

### 3. DEV URN Definition

Namespace Identifier: "dev" requested

Version: 1

Date: 2018-03-19

Registration Information: This is the first registration of this namespace, 2018-03-19.

Registrant: IETF and the CORE working group. Should the working group cease to exist, discussion should be directed to the general IETF discussion forums or the IESG.

#### 3.1. Purpose

Purpose: The DEV URNs identify devices with device-specific identifiers such as network card hardware addresses. These URNs may be used in any relevant networks that benefit from the ability to refer to these identifiers in the form of URNs; DEV URN is global in scope.

Some typical applications include equipment inventories and smart object systems.

DEV URNs can be used in various ways in applications, software systems, and network components, in tasks ranging from discovery (for instance when discovering 1-wire network devices or detecting MAC-addressable devices on a LAN) to intrusion detection systems and simple catalogues of system information.

While it is possible to implement resolution systems for specific applications or network locations, DEV URNs are typically not used in a way that requires resolution beyond direct observation of the relevant identity fields in local link communication. However, it is often useful to be able to pass device identity information in generic URN fields in databases or protocol fields, which makes the use of URNs for this purpose convenient.

The DEV URN name space complements existing name spaces such as those involving IMEI or UUID identifiers. DEV URNs are expected to be a part of the IETF-provided basic URN types, covering identifiers that have previously not been possible to use in URNs.

### 3.2. Syntax

**Syntax:** The identifier is expressed in ASCII characters and has a hierarchical structure as follows:

```
devurn = "urn:dev:" body componentpart
body = macbody / owbody / orgbody / osbody / opsboddy / otherbody
macbody = "mac:" hexstring
owbody = "ow:" hexstring
orgbody = "org:" number "-" identifier
osbody = "os:" number "-" serial
opsbody = "ops:" number "-" product "-" serial
otherbody = subtype ":" identifier
subtype = ALPHA *(DIGIT / ALPHA)
identifier = 1*unreservednout
product = identifier
serial = identifier
unreservednout = ALPHA / DIGIT / "_"
componentpart = [ "_" component [ componentpart ] ]
component = *1(DIGIT / ALPHA)
hexstring = hexbyte /
             hexbyte hexstring
hexbyte = hexdigit hexdigit
hexdigit = DIGIT / hexletter
hexletter = "a" / "b" / "c" / "d" / "e" / "f"
number = *1DIGIT
```

The above Augmented Backus-Naur Form (ABNF) uses the DIGIT and ALPHA rules defined in [RFC5234], which are not repeated here. The rule for pct-encoding is defined in Section 2.1 of [RFC3986].

The device identity namespace includes three subtypes (see Section 4, and more may be defined in the future as specified in Section 7.

The optional components following the hexstring are strings depicting individual aspects of a device. The specific strings and their semantics are up to the designers of the device, but could be used to refer to specific interfaces or functions within the device.

There are no special character encoding rules or considerations for conforming with the URN syntax, beyond those applicable for URNs in general [RFC8141], or the context where these URNs are carried (e.g., inside JSON [RFC8259] or SenML [RFC8428]).

The lexical equivalence of the DEV URNs is defined as an exact and case sensitive string match. Note that the two subtypes defined in this document use only lower case letters, however. Future types

might use identifiers that require other encodings that require a more full-blown character set (such as BASE64), however.

DEV URNs do not use r-, q-, or f-components.

Specific subtypes of DEV URNs may be validated through mechanisms discussed in Section 4.

Finally, the string representation of the device identity URN and of the MEID sub namespace is fully compatible with the URN syntax.

### 3.3. Assignment

**Assignment:** The process for identifier assignment is dependent on the used subtype, and documented in the specific subsection under Section 4.

Device identifiers are generally expected to be unique, barring the accidental issue of multiple devices with the same identifiers.

This URN type SHOULD only be used for persistent identifiers, such as hardware-based identifiers or cryptographic identifiers based on keys intended for long-term usage.

### 3.4. Security and Privacy

**Security and Privacy:** As discussed in Section 6, care must be taken to use device identifier-based identifiers due to their nature as a long-term identifier that is often not changeable. Leakage of these identifiers outside systems where their use is justified should be controlled.

### 3.5. Interoperability

**Interoperability:** There are no specific interoperability concerns.

### 3.6. Resolution

**Resolution:** The device identities are not expected to be globally resolvable. No identity resolution system is expected. Systems may perform local matching of identities to previously seen identities or configured information, however.

### 3.7. Documentation

See RFC NNNN (RFC Editor: Please replace NNNN by a reference to the RFC number of this document).

### 3.8. Additional Information

See Section 1 for a discussion of related name spaces.

### 3.9. Revision Information

Revision Information: This is the first version of this registration.

## 4. DEV URN Subtypes

### 4.1. MAC Addresses

DEV URNs of the "mac" subtype are based on the EUI-64 identifier [IEEE.EUI64] derived from a device with a built-in 64-bit EUI-64. The EUI-64 is formed from 24 or 36 bits of organization identifier followed by 40 or 28 bits of device-specific extension identifier assigned by that organization.

In the DEV URN "mac" subtype the hexstring is simply the full EUI-64 identifier represented as a hexadecimal string. It is always exactly 16 characters long.

MAC-48 and EUI-48 identifiers are also supported by the same DEV URN subtype. To convert a MAC-48 address to an EUI-64 identifier, The OUI of the Ethernet address (the first three octets) becomes the organization identifier of the EUI-64 (the first three octets). The fourth and fifth octets of the EUI are set to the fixed value FFFF hexadecimal. The last three octets of the Ethernet address become the last three octets of the EUI-64. The same process is used to convert an EUI-48 identifier, but the fixed value FFFE is used instead.

Identifier assignment for all of these identifiers rests within the IEEE.

### 4.2. 1-Wire Device Identifiers

The 1-Wire\* system is a device communications bus system designed by Dallas Semiconductor Corporation. 1-Wire devices are identified by a 64-bit identifier that consists of 8 byte family code, 48 bit identifier unique within a family, and 8 bit CRC code [OW].

\*) 1-Wire is a registered trademark.

In DEV URNs with the "ow" subtype the hexstring is a representation of the full 64 bit identifier as a hexadecimal string. It is always exactly 16 characters long. Note that the last two characters

represent the 8-bit CRC code. Implementations MAY check the validity of this code.

Family code and identifier assignment for all 1-wire devices rests with the manufacturers.

#### 4.3. Organization-Defined Identifiers

Device identifiers that have only a meaning within an organisation can also be used to represent vendor-specific or experimental identifiers or identifiers designed for use within the context of an organisation. Organisations are identified by their Private Enterprise Number (PEN) [RFC2578].

#### 4.4. Organization Serial Numbers

The "os" subtype specifies an organization and a serial number. Organizations are identified by their PEN.

Note: The DEV URN "os" subtype has originally been defined in the LwM2M standard, but has been incorporated here to collect all syntax associated with DEV URNs in one place. At the same time, the syntax of this subtype was changed to avoid the possibility of characters that are not allowed in SenML Name field (see [RFC8428] Section 4.5.1).

#### 4.5. Organization Product and Serial Numbers

The DEV URN "ops" subtype has originally been defined in the LwM2M standard, but has been incorporated here to collect all syntax associated with DEV URNs in one place. The "ops" subtype specifies an organization, product class, and a serial number. Organizations are identified by their PEN.

Note: As with the "os" subtype, the "ops" subtype has originally been defined in the LwM2M standard, and its format has been slightly changed.

#### 5. Examples

The following three examples provide examples of MAC-based, 1-Wire, and Cryptographic identifiers:

```
urn:dev:mac:0024beffffe804ff1      # The MAC address of
                                     # Jari's laptop

urn:dev:ow:10e2073a01080063         # The 1-Wire temperature
                                     # sensor in Jari's
                                     # kitchen

urn:dev:ow:264437f5000000ed_humidity # The laundry sensor's
                                     # humidity part

urn:dev:ow:264437f5000000ed_temperature # The laundry sensor's
                                     # temperature part

urn:dev:org:32473-123456             # Device 123456 in
                                     # the RFC 5612 example
                                     # organisation

urn:dev:ops:32473-Refrigerator-5002  # Refrigerator serial
                                     # number 5002 in the
                                     # RFC 5612 example
                                     # organisation
```

## 6. Security Considerations

On most devices, the user can display device identifiers. Depending on circumstances, device identifiers may or may not be modified or tampered by the user. An implementation of the DEV URN MUST NOT change these properties from what they were intended. In particular, a device identifier that is intended to be immutable should not become mutable as a part of implementing the DEV URN type. More generally, nothing in this memo should be construed to override what the relevant device specifications have already said about the identifiers.

Other devices in the same network may or may not be able to identify the device. For instance, on Ethernet network, the MAC address of a device is visible to all other devices.

The URNs generated according to the rules defined in this document result in long-term stable unique identifiers for the devices. Such identifiers may have privacy and security implications because they may enable correlating information about a specific device over a long period of time, location tracking, and device specific vulnerability exploitation [RFC7721]. Also, usually there is no easy way to change the identifier. Therefore these identifiers need to be used with care and especially care should be taken avoid leaking them outside of the system that is intended to use the identifiers.



## 7. IANA Considerations

This document requests the registration of a new URN namespace for "DEV", as described in Section 3.

Additional subtypes for DEV URNs can be defined through IETF Review or IESG Approval [RFC5226].

Such allocations are appropriate when there is a new namespace of some type of device identifiers, defined in stable fashion and with a publicly available specification that can be pointed to.

Note that the organisation (Section 4.3) device identifiers can also be used in some cases, at least as a temporary measure. It is preferable, however, that long-term usage of a broadly employed device identifier be registered with IETF rather than used through the organisation device identifier type.

## 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>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<https://www.rfc-editor.org/info/rfc2578>>.
- [RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", RFC 3406, DOI 10.17487/RFC3406, October 2002, <<https://www.rfc-editor.org/info/rfc3406>>.
- [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>>.

- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <<https://www.rfc-editor.org/info/rfc5226>>.
- [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>>.
- [IEEE.EUI64] IEEE, "Guidelines For 64-bit Global Identifier (EUI-64)", IEEE , unknown year, <<http://standards.ieee.org/db/oui/tutorials/EUI64.html>>.
- [OW] IEEE, "Overview of 1-Wire(R) Technology and Its Use", MAXIM <http://www.maxim-ic.com/app-notes/index.mvp/id/1796>, June 2008, <<http://www.maxim-ic.com/app-notes/index.mvp/id/1796>>.

## 8.2. Informative References

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<https://www.rfc-editor.org/info/rfc2616>>.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <<https://www.rfc-editor.org/info/rfc4627>>.

- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<https://www.rfc-editor.org/info/rfc7721>>.
- [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>>.
- [W3C.REC-xml-19980210] Sperberg-McQueen, C., Bray, T., and J. Paoli, "XML 1.0 Recommendation", World Wide Web Consortium FirstEdition REC-xml-19980210, February 1998, <<http://www.w3.org/TR/1998/REC-xml-19980210>>.
- [OUI] IEEE, SA., "Registration Authority", IEEE-SA webpage, 2018, <<http://standards.ieee.org/develop/regauth/oui/>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC7254] Montemurro, M., Ed., Allen, A., McDonald, D., and P. Gosden, "A Uniform Resource Name Namespace for the Global System for Mobile Communications Association (GSMA) and the International Mobile station Equipment Identity (IMEI)", RFC 7254, DOI 10.17487/RFC7254, May 2014, <<https://www.rfc-editor.org/info/rfc7254>>.
- [I-D.atarius-dispatch-meid-urn] Atarius, R., "A Uniform Resource Name Namespace for the Device Identity and the Mobile Equipment Identity (MEID)", draft-atarius-dispatch-meid-urn-18 (work in progress), June 2018.

## Appendix A. Changes from Previous Version

Version -03 of the WG draft removed some unnecessary references, updated some other references, removed pct-encoding to ensure the DEV URNs fit [RFC8428] Section 4.5.1 rules, and clarified that the original source of the "os" and "ops" subtypes.

Version -02 of the WG draft folded in the "ops" and "os" branches of the dev:urn syntax from LwM2M, as they seemed to match well what already existed in this memo under the "org" branch. However, as a part of this three changes were incorporated:

- o The syntax for the "org:" changes to use "-" rather than ":" between the OUI and the rest of the URN.
- o The organizations for the "ops" and "os" branches have been changed to use PEN numbers rather than OUI numbers [OUI]. The reason for this is that PEN numbers are allocated through a simpler and less costly process. However, this is a significant change to how LwM2M identifiers were specified before.
- o There were also changes to what general characters can be used in the otherbody branch of the ABNF.

The rationale for all these changes is that it would be helpful for the community collect and unify syntax between the different uses of DEV URNs. If there is significant use of either the org:, os:, or ops: subtypes, then changes at this point may not be warranted, but otherwise unified syntax, as well as the use of PEN numbers would probably be beneficial. Comments on this topic are appreciated.

Version -01 of the WG draft converted the draft to use the new URN registration template from [RFC8141].

Version -00 of the WG draft renamed the file name and fixed the ABNF to correctly use "org:" rather than "dn:".

Version -05 made a change to the delimiter for parameters within a DEV URN. Given discussions on allowed character sets in SenML [RFC8428], we would like to suggest that the "\_" character be used instead of ";", to avoid the need to translate DEV URNs in SenML-formatted communications or files. However, this reverses the earlier decision to not use unreserved characters. This also means that device IDs cannot use "\_" characters, and have to employ other characters instead. Feedback on this decision is sought.

Version -05 also introduced local or organisation-specific device identifiers. Organisations are identified by their PEN number

(although we considered FQDNs as a potential alternative. The authors believe an organisation-specific device identifier type will make experiments and local use easier, but feedback on this point and the choice of PEN numbers vs. other possible organisation identifiers would be very welcome.

Version -05 also added some discussion of privacy concerns around long-term stable identifiers.

Finally, version -05 clarified the situations when new allocations within the registry of possible device identifier subtypes is appropriate.

Version -04 is a refresh, as the need and interest for this specification has re-emerged. And the editing author has emerged back to actual engineering from the depths of IETF administration.

Version -02 introduced several changes. The biggest change is that with the NI URNs [RFC6920], it was no longer necessary to define cryptographic identifiers in this specification. Another change was that we incorporated a more generic syntax for future extensions; non-hexstring identifiers can now also be supported, if some future device identifiers for some reason would, for instance, use BASE64. As a part of this change, we also changed the component part separator character from '-' to ';' so that the general format of the rest of the URN can employ the unreserved characters [RFC3986].

## Appendix B. Acknowledgments

The authors would like to thank Ari Keranen, Stephen Farrell, Christer Holmberg, Peter Saint-Andre, Wouter Cloetens, Jaime Jimenez, Padmakumar Subramani, Mert Ocak, Hannes Tschofenig, and Ahmad Muhanna for interesting discussions in this problem space. We would also like to note prior documents that focused on specific device identifiers, such as [RFC7254] or [I-D.atarius-dispatch-meid-urn].

## Authors' Addresses

Jari Arkko  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net

Cullen Jennings  
Cisco  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Phone: +1 408 421-9990  
Email: fluffy@cisco.com

Zach Shelby  
ARM  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: Zach.Shelby@arm.com

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 9, 2019

Z. Shelby  
ARM  
M. Koster  
SmartThings  
C. Groves

J. Zhu  
Huawei  
B. Silverajan, Ed.  
Tampere University  
March 08, 2019

Dynamic Resource Linking for Constrained RESTful Environments  
draft-ietf-core-dynlink-08

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252). This specification also defines Conditional Notification Attributes that work with Link Bindings or with CoAP Observe (RFC7641).

Editor note

The git repository for the draft is found at <https://github.com/core-wg/dynlink>

Status of This Memo

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

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

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

This Internet-Draft will expire on September 9, 2019.

## Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Conditional Notification Attributes . . . . .	4
3.1. Attribute Definitions . . . . .	4
3.1.1. Minimum Period (pmin) . . . . .	5
3.1.2. Maximum Period (pmax) . . . . .	5
3.1.3. Change Step (st) . . . . .	5
3.1.4. Greater Than (gt) . . . . .	6
3.1.5. Less Than (lt) . . . . .	6
3.1.6. Notification Band (band) . . . . .	7
3.2. Server processing of Conditional Notification Attributes . . . . .	8
4. Link Bindings . . . . .	8
4.1. The "bind" attribute and Binding Methods . . . . .	9
4.1.1. Polling . . . . .	10
4.1.2. Observe . . . . .	10
4.1.3. Push . . . . .	11
4.2. Link Relation . . . . .	11
5. Binding Table . . . . .	11
6. Implementation Considerations . . . . .	12
7. Security Considerations . . . . .	13
8. IANA Considerations . . . . .	13
8.1. Resource Type value 'core.bnd' . . . . .	13
8.2. Link Relation Type . . . . .	13
9. Acknowledgements . . . . .	14
10. Contributors . . . . .	14
11. Changelog . . . . .	14
12. References . . . . .	16
12.1. Normative References . . . . .	16
12.2. Informative References . . . . .	17
Appendix A. Examples . . . . .	17
A.1. Greater Than (gt) example . . . . .	17



A.2. Greater Than (gt) and Period Max (pmax) example . . . . .	18
Authors' Addresses . . . . .	19

## 1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol [RFC7252] and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format [RFC6690] is a standard for doing Web Linking [RFC8288] in constrained environments.

This specification introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which state updates are conveyed. Specifically, a Link Binding is a unidirectional link for binding the states of source and destination resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This specification additionally defines Conditional Notification Attributes for use with Link Bindings and with the CoRE Observe [RFC7641] method.

## 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 all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This specification makes use of the following additional terminology:

**Link Binding:** A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

**State Synchronization:** Depending on the binding method (Polling, Observe, Push) different REST methods may be used to synchronize the resource values between a source and a destination. The process of using a REST method to achieve this is defined as "State Synchronization". The endpoint triggering the state synchronization is the synchronization initiator.

**Notification Band:** A resource value range that results in state synchronization. The value range may be bounded by a minimum and

maximum value or may be unbounded having either a minimum or maximum value.

### 3. Conditional Notification Attributes

#### 3.1. Attribute Definitions

This specification defines Conditional Notification Attributes, which provide for fine-grained control of notification and state synchronization when using CoRE Observe [RFC7641] or Link Bindings (see Section 4). Conditional Notification Attributes define the conditions that trigger a notification.

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY also be used to observe any changes in a resource, and receive asynchronous notifications as a result. A resource marked as Observable in its link description SHOULD support these Conditional Notification Attributes.

The set of parameters defined here allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting.

One or more Notification Attributes MAY be included as query parameters in an Observe request.

These attributes are defined below:

Attribute	Parameter	Value
Minimum Period (s)	pmin	xsd:decimal (>0)
Maximum Period (s)	pmax	xsd:decimal (>0)
Change Step	st	xsd:decimal (>0)
Greater Than	gt	xsd:decimal
Less Than	lt	xsd:decimal
Notification Band	band	xsd:boolean

Table 1: Conditional Notification Attributes

Conditional Notification Attributes SHOULD be evaluated on all potential notifications from a resource, whether resulting from an internal server-driven sampling process or from external update requests to the server.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value of a resource; specifically, that a resource may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the Conditional Notification Attributes and instead use the phrase "sampled value" to refer to a member of a sequence of values that may be internally observed from the resource state over time.

#### 3.1.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the minimum period is up to the server. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

A server MAY report the last sampled value that occurred during the pmin interval, after the pmin interval expires.

Note: Due to finite quantization effects, the time between notifications may be greater than pmin even when the sampled value changes within the pmin interval. Pmin may or may not be used to drive the internal sampling process.

#### 3.1.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than zero and MUST be greater than the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

#### 3.1.3. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a notification, compared to the value of the previous notification. Upon reception of a query including the st attribute, the most recently sampled value of the resource is reported, and then set as the last reported value

(last\_rep\_v). When a subsequent sample or update of the resource value differs from the last reported value by an amount, positive or negative, greater than or equal to st, and the time for pmin has elapsed since the last notification, a notification is sent and the last reported value is updated to the value sent in the notification. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

The Change Step parameter can only be supported on resources with a scalar numeric value.

Note: Due to sampling and other constraints, e.g. pmin, the resource value received in two sequential notifications may differ by more than st.

#### 3.1.4. Greater Than (gt)

When present, Greater Than indicates the upper limit value the sampled value SHOULD cross before triggering a notification. A notification is sent whenever the sampled value crosses the specified upper limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to rise, no notifications are generated as a result of gt. If the value drops below the upper limit value then a notification is sent, subject again to the pmin time.

The Greater Than parameter can only be supported on resources with a scalar numeric value.

#### 3.1.5. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a notification. A notification is sent when the sampled value crosses the specified lower limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to fall no notifications are generated as a result of lt. If the value rises above the lower limit value then a new notification is sent, subject to the pmin time..

The Less Than parameter can only be supported on resources with a scalar numeric value.

### 3.1.6. Notification Band (band)

The notification band attribute allows a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple notifications. This enables use cases where different ranges results in differing behaviour. For example: monitoring the temperature of machinery. Whilst the temperature is in the normal operating range only periodic observations are needed. However as the temperature moves to more abnormal ranges more frequent synchronization/reporting may be needed.

Without a notification band, a transition across a less than (lt), or greater than (gt) limit only generates one notification. This means that it is not possible to describe a case where multiple notifications are sent so long as the limit is exceeded.

The band attribute works as a modifier to the behaviour of gt and lt. Therefore, if band is present in a query, gt, lt or both, MUST be included.

When band is present with the lt attribute, it defines the lower bound for the notification band (notification band minimum). Notifications occur when the resource value is equal to or above the notification band minimum. If lt is not present there is no minimum value for the band.

When band is present with the gt attribute, it defines the upper bound for the notification band (notification band maximum). Notifications occur when the resource value is equal to or below the notification band maximum. If gt is not present there is no maximum value for the band.

If band is present with both the gt and lt attributes, notification occurs when the resource value is greater than or equal to gt or when the resource value is less than or equal to lt.

If a band is specified in which the value of gt is less than that of lt, in-band notification occurs. That is, notification occurs whenever the resource value is between the gt and lt values, including equal to gt or lt.

If the band is specified in which the value of gt is greater than that of lt, out-of-band notification occurs. That is, notification occurs when the resource value not between the gt and lt values, excluding equal to gt and lt.

The Notification Band parameter can only be supported on resources with a scalar numeric value.

### 3.2. Server processing of Conditional Notification Attributes

Pmin, pmax, st, gt, lt and band may be present in the same query. However, they are not defined at multiple prioritization levels. The server sends a notification whenever any of the parameter conditions are met, upon which it updates its last notification value and time to prepare for the next notification. Only one notification occurs when there are multiple conditions being met at the same time. The reference code below illustrates the logic to determine when a notification is to be sent.

```
bool notifiable( Resource * r ) {

#define BAND r->band
#define SCALAR_TYPE ( num_type == r->type )
#define STRING_TYPE ( str_type == r->type )
#define BOOLEAN_TYPE ( bool_type == r->type )
#define PMIN_EX ( r->last_sample_time - r->last_rep_time >= r->pmin )
#define PMAX_EX ( r->last_sample_time - r->last_rep_time > r->pmax )
#define LT_EX ( r->v < r->lt ^ r->last_rep_v < r->lt )
#define GT_EX ( r->v > r->gt ^ r->last_rep_v > r->gt )
#define ST_EX ( abs( r->v - r->last_rep_v ) >= r->st )
#define IN_BAND ( ( r->gt <= r->v && r->v <= r->lt ) || ( r->lt <= r->gt && r->g
t <= r->v ) || ( r->v <= r->lt && r->lt <= r->gt ) )
#define VB_CHANGE ( r->vb != r->last_rep_vb )
#define VS_CHANGE ( r->vs != r->last_rep_vs )

    return (
        PMIN_EX &&
        ( SCALAR_TYPE ?
            ( ( !BAND && ( GT_EX || LT_EX || ST_EX || PMAX_EX ) ) ||
              ( BAND && IN_BAND && ( ST_EX || PMAX_EX ) ) )
        : STRING_TYPE ?
            ( VS_CHANGE || PMAX_EX )
        : BOOLEAN_TYPE ?
            ( VB_CHANGE || PMAX_EX )
        : false )
    );
}
```

Figure 1: Code logic for conditional notification attribute interactions

### 4. Link Bindings

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control.

Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool.

In this specification such an abstract relationship between two resources is defined, called a Link Binding. The configuration phase necessitates the exchange of binding information, so a format recognized by all CoRE endpoints is essential. This specification defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources.

The purpose of such a binding is to synchronize content updates between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method.

Conditional Notification Attributes defined in Section 3 can be used with Link Bindings in order to customize the notification behavior and timing.

#### 4.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Attribute	Parameter	Value
Binding method	bind	xsd:string

Table 2: The bind attribute

The following table gives a summary of the binding methods defined in this specification.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

Table 3: Binding Method Summary

The description of a binding method defines the following aspects:

**Identifier:** This is the value of the "bind" attribute used to identify the method.

**Location:** This information indicates whether the binding entry is stored on the source or on the destination endpoint.

**REST Method:** This is the REST method used in the Request/Response exchanges.

**Conditional Notification:** How Conditional Notification Attributes are used in the binding.

The binding methods are described in more detail below.

#### 4.1.1. Polling

The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method **MUST** be stored on the destination endpoint. The destination endpoint **MUST** ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process **MAY** filter out content from the GET requests using value-based conditions (e.g based on the Change Step, Less Than, Greater Than attributes).

#### 4.1.2. Observe

The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [RFC7641] hence this method is only permitted when the resources are made available over CoAP. The



binding entry for this method **MUST** be stored on the destination endpoint. The binding conditions are mapped as query parameters in the Observe request (see Section 3).

#### 4.1.3. Push

When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint **SHOULD** only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method **MUST** be stored on the source endpoint.

#### 4.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource.

### 5. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource **SHOULD** indicate its presence and enable its discovery by advertising a link at `"/.well-known/core"` [RFC6690]. If so, the Binding Table resource **MUST** be discoverable by using the Resource Type (rt) `'core.bnd'`.

The Methods column defines the REST methods supported by the Binding Table, which are described in more detail below.

Resource	rt=	Methods	Content-Format
Binding Table	core.bnd	GET, PUT	link-format

Table 4: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of `application/link-format` is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT request **MUST** have a relation type "boundto".

(Editor's Note: Usage of the PATCH method for fine-grained addition and removal of individual bindings is under study.)

The following example shows requests for discovering, retrieving and replacing bindings in a binding table.

```
Req: GET /.well-known/core?rt=core.bnd (application/link-format)
Res: 2.05 Content (application/link-format)
</bnd/>;rt=core.bnd;ct=40
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/a/switch1/>;
    rel=boundto;bind=obs;anchor=/a/fan;bind="obs",
<coap://sensor.example.com/a/switch2/>;
    rel=boundto;bind=obs;anchor=/a/light;bind="obs"
```

```
Req: PUT /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
```

Figure 2: Binding Table Example

## 6. Implementation Considerations

When using multiple resource bindings (e.g. multiple Observations of resource) with different bands, consideration should be given to the resolution of the resource value when setting sequential bands. For example: Given BandA (Abmn=10, BbmX=20) and BandB (Bbmn=21, BbmX=30). If the resource value returns an integer then notifications for values between and inclusive of 10 and 30 will be triggered. Whereas if the resolution is to one decimal point (0.1) then notifications for values 20.1 to 20.9 will not be triggered.

The use of the notification band minimum and maximum allow for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period for the determining the resource value. Implementors SHOULD consider the resolution needed before updating the resource, e.g. updating the resource when a temperature sensor value changes by 0.001 degree versus 1 degree.

The initiation of a Link Binding can be delegated from a client to a link state machine implementation, which can be an embedded client or a configuration tool. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

## 7. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

## 8. IANA Considerations

### 8.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 5. This attribute value is used to discover the resource representing a binding table, which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

### 8.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC8288].

Relation Name: boundto

Description: The purpose of a boundto relation type is to indicate that there is a binding between a source resource and a destination resource for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

Application Data: None

## 9. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this specification. Christian Amsuss supplied a comprehensive review of draft -06.

## 10. Contributors

Matthieu Vial  
Schneider-Electric  
Grenoble  
France

Phone: +33 (0)47657 6522  
EMail: matthieu.vial@schneider-electric.com

## 11. Changelog

draft-ietf-core-dynlink-08

- o Reorganize the draft to introduce Conditional Notification Attributes at the beginning
- o Made pmin and pmax type xsd:decimal to accommodate fractional second timing
- o updated the attribute descriptions. lt and gt notify on all crossings, both directions
- o updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- o Added reference code to illustrate attribute interactions for observations

## draft-ietf-core-dynlink-06

- o Document restructure and refactoring into three main sections
- o Clarifications on band usage
- o Implementation considerations introduced
- o Additional text on security considerations

## draft-ietf-core-dynlink-05

- o Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- o Removed statement prescribing gt MUST be greater than lt

## draft-ietf-core-dynlink-03

- o General: Reverted to using "gt" and "lt" from "gth" and "lth" for this draft owing to concerns raised that the attributes are already used in LwM2M with the original names "gt" and "lt".
- o New author and editor added.

## draft-ietf-core-dynlink-02

- o General: Changed the name of the greater than attribute "gt" to "gth" and the name of the less than attribute "lt" to "lth" due to conflict with the core resource directory draft lifetime "lt" attribute.
- o Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".
- o Added Appendix A for examples.

## draft-ietf-core-dynlink-01

- o General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- o General: The document has been restructured to make the information flow better.
- o Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.

- o Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- o Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- o Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- o Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- o This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- o This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- o The WADL description has been dropped in favour of a thorough textual description of the REST API.

## 12. References

### 12.1. 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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

## 12.2. Informative References

- [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>>.

## Appendix A. Examples

This appendix provides some examples of the use of binding attribute / observe attributes.

Note: For brevity the only the method or response code is shown in the header field.

### A.1. Greater Than (gt) example

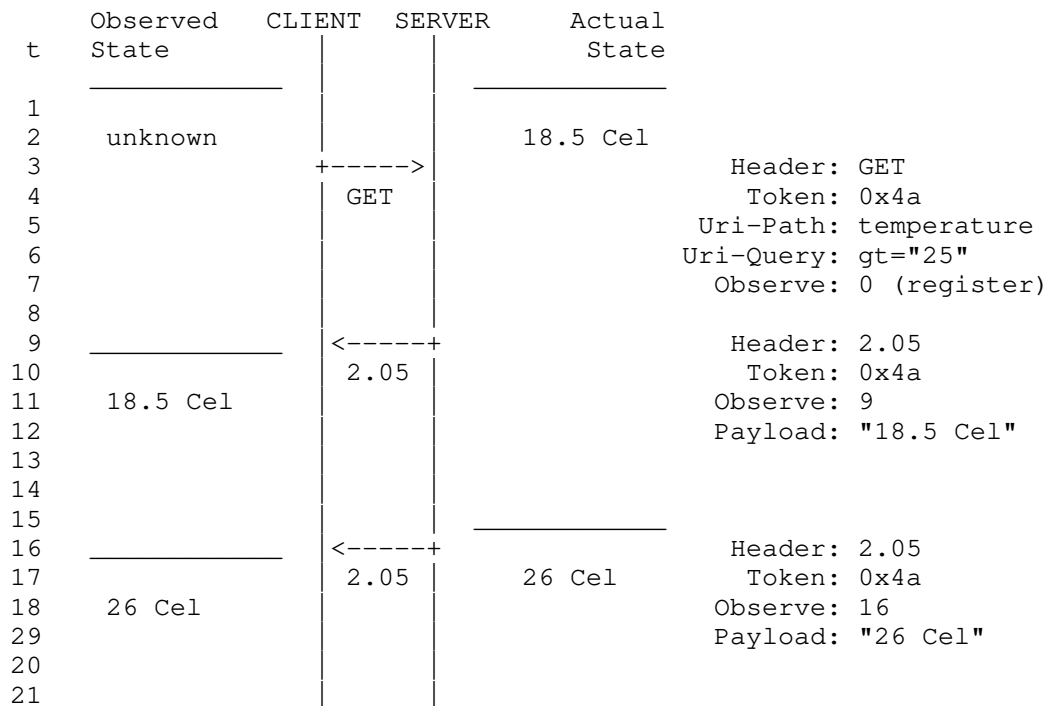
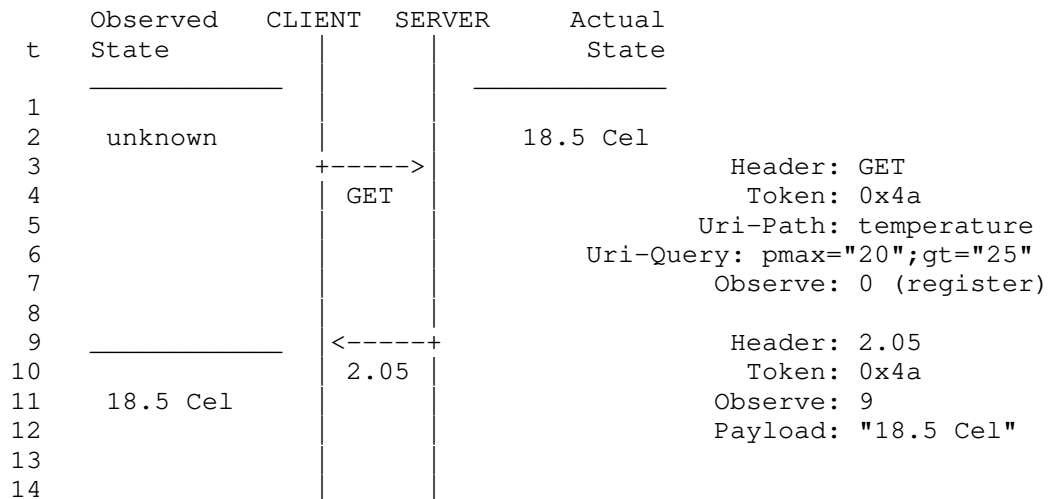


Figure 3: Client Registers and Receives one Notification of the Current State and One of a New State when it passes through the greather than threshold of 25.

#### A.2. Greater Than (gt) and Period Max (pmax) example





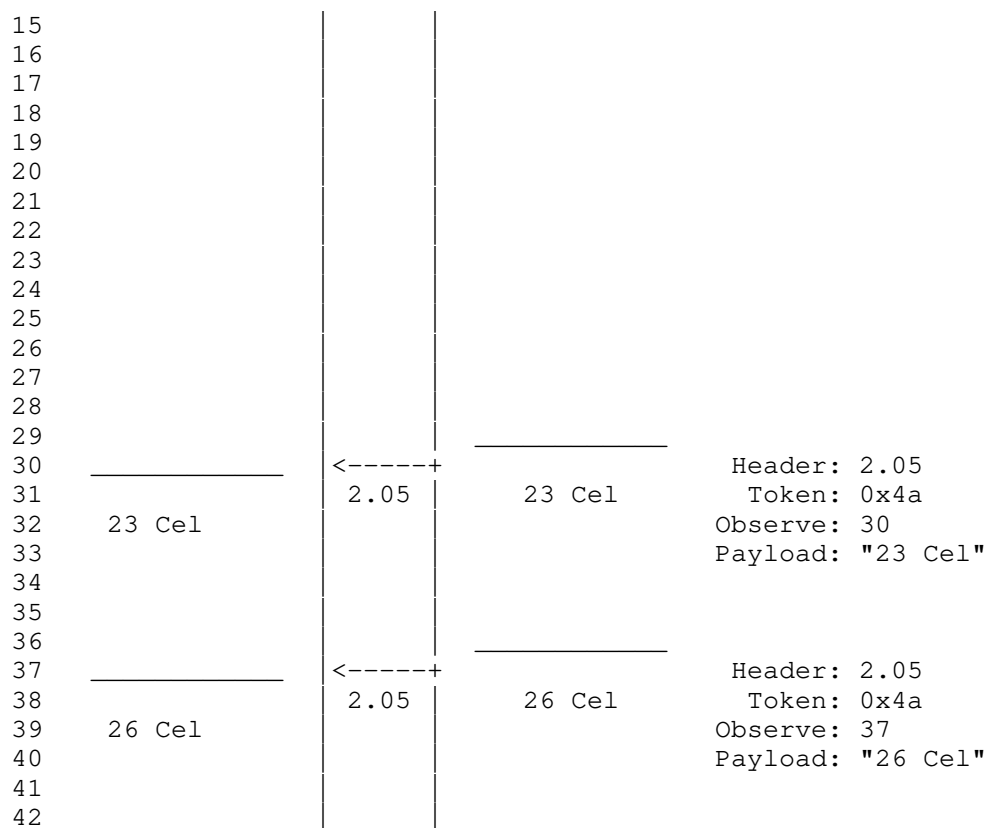


Figure 4: Client Registers and Receives one Notification of the Current State, one when pmax time expires and one of a new State when it passes through the greather than threshold of 25.

#### Authors' Addresses

Zach Shelby  
ARM  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach.shelby@arm.com

Michael Koster  
SmartThings  
665 Clyde Avenue  
Mountain View 94043  
USA

Email: michael.koster@smarththings.com

Christian Groves  
Australia

Email: cngroves.std@gmail.com

Jintao Zhu  
Huawei  
No.127 Jinye Road, Huawei Base, High-Tech Development District  
Xi'an, Shaanxi Province  
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)  
Tampere University  
Kalevantie 4  
Tampere FI-33100  
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group  
Internet-Draft  
Updates: 7252 (if approved)  
Intended status: Standards Track  
Expires: April 25, 2019

C. Amsuess  
J. Mattsson  
G. Selander  
Ericsson AB  
October 22, 2018

Echo and Request-Tag  
draft-ietf-core-echo-request-tag-03

Abstract

This document specifies security enhancements to the Constrained Application Protocol (CoAP). Two optional extensions are defined: the Echo option and the Request-Tag option. Each of these options provide additional features to CoAP and protects against certain attacks. The document also updates the processing requirements on the Token of RFC 7252. The updated Token processing ensures secure binding of responses to requests.

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 25, 2019.

Copyright Notice

Copyright (c) 2018 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. Request Freshness . . . . .	3
1.2. Fragmented Message Body Integrity . . . . .	4
1.3. Request-Response Binding . . . . .	4
1.4. Terminology . . . . .	5
2. The Echo Option . . . . .	6
2.1. Option Format . . . . .	6
2.2. Echo Processing . . . . .	7
2.3. Applications . . . . .	9
3. The Request-Tag Option . . . . .	11
3.1. Option Format . . . . .	11
3.2. Request-Tag Processing by Servers . . . . .	12
3.3. Setting the Request-Tag . . . . .	13
3.4. Applications . . . . .	13
3.4.1. Body Integrity Based on Payload Integrity . . . . .	13
3.4.2. Multiple Concurrent Blockwise Operations . . . . .	14
3.4.3. Simplified Block-Wise Handling for Constrained Proxies . . . . .	15
3.5. Rationale for the Option Properties . . . . .	15
3.6. Rationale for Introducing the Option . . . . .	16
4. Block2 / ETag Processing . . . . .	16
5. Token Processing . . . . .	16
6. Security Considerations . . . . .	16
7. Privacy Considerations . . . . .	17
8. IANA Considerations . . . . .	18
9. References . . . . .	18
9.1. Normative References . . . . .	18
9.2. Informative References . . . . .	18
Appendix A. Methods for Generating Echo Option Values . . . . .	20
Appendix B. Request-Tag Message Size Impact . . . . .	21
Appendix C. Change Log . . . . .	21
Acknowledgments . . . . .	22
Authors' Addresses . . . . .	22

## 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 server-oriented 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 apparent 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 blockwise 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 ([I-D.ietf-core-object-security]), provide the additional security features.

The document also updates the processing requirements on the Token. The updated processing ensures secure binding of responses to requests, thus mitigating error cases and attacks where the client may erroneously associate the wrong response to a request.

### 1.1. 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 incriminating in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks are possible to mitigate by establishing fresh session keys, e.g. performing a DTLS handshake for each actuation, 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. The Echo option, defined in this document, specifies such a mechanism which thereby enables a CoAP server to verify the freshness of a request. 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 and to verify aliveness of the client.

## 1.2. 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 blockwise 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. This document defines the Request-Tag option for identifying the request body fragmented using the Block1 option, similar to ETag, but ephemeral and set by the CoAP client.

## 1.3. 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 valid 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, binding is 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. Concurrent requests are instead 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].

Unfortunately, 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 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 never reuse tokens until the AEAD keys have been replaced. As there may be any number of responses to a request (see e.g. [RFC7641]), the easiest way to accomplish this is to implement the token as a counter and never reuse any tokens at all. This document updates the Token processing in [RFC7252] to always assure a cryptographically secure binding of responses to requests.

#### 1.4. 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 "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise 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 blockwise 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 blockwise request operations are impossible with the options of [RFC7959] 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. The Echo Option

The Echo option is a lightweight server-driven challenge-response mechanism for CoAP, motivated by the need for a server to verify freshness of a request as described in Section 1.1. With request freshness we mean that the server can determine that the client (or in the case of hop-by-hop security the proxy) sent the request recently. The time threshold for being fresh is application specific. 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.

### 2.1. 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

[ Note to RFC editor: If this document is released before core-object-security, then the following paragraph and the "E"/"U" columns above need to move into core-object-security, as they are defined in that draft. ]

The Echo option MAY be an Inner or Outer option [I-D.ietf-core-object-security], and the Inner and Outer values are



independent. 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.

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 was generated by the server as well as the point in time when the Echo option value was generated.

## 2.2. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.3 for different applications), but the Echo option MUST NOT be used with empty CoAP requests (i.e. Code=0.00).

If a server receives a request which has freshness requirements, 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 responses and to different clients.

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.

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 their apparent network address.

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.

Upon receiving a request with the Echo option, the server determines if the request has freshness requirements. If the request does not have freshness requirements, the Echo option MAY be ignored. If the request has freshness requirements and the server cannot verify the freshness of the request in some other way, the server MUST verify that the Echo option value was generated by the server; otherwise the request is not processed further. The server MUST then calculate the round-trip time  $RTT = (t1 - t0)$ , where  $t1$  is the request receive time and  $t0$  is the time when the Echo option value was generated. The server MUST only accept requests with a round-trip time below a certain threshold  $T$ , i.e.  $RTT < T$ . If the server cannot verify that the Echo option value was generated by the server or the round-trip time is not below the threshold the request is not processed further, and an error message MAY be sent. The error message SHOULD include a new Echo option. The threshold  $T$  is application specific, its value depends e.g. on the freshness requirements of the request. An example message flow is illustrated in Figure 2.

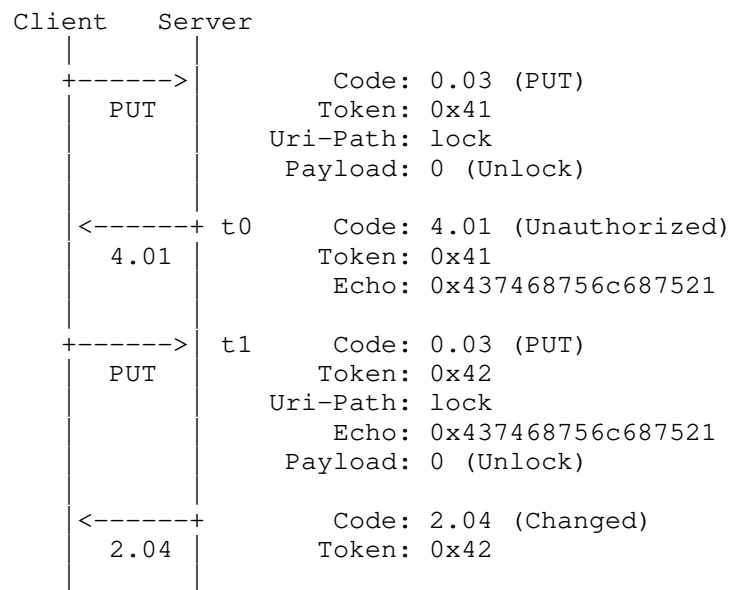


Figure 2: Example Echo Option Message Flow

Note that the server does not have to synchronize the time used for the Echo timestamps with any other party. However, if the server loses time continuity, e.g. due to reboot, it MUST reject all Echo values that was created before time continuity was lost.

When used to serve freshness requirements (including client aliveness and state synchronizing), CoAP messages containing the Echo option MUST be integrity protected between the intended endpoints, e.g. using DTLS, TLS, or an OSCORE Inner option ([I-D.ietf-core-object-security]). When used to demonstrate reachability at their apparent network address, the Echo option MAY be unprotected.

A CoAP-to-CoAP proxy MAY respond to requests with 4.01 with an Echo option to ensure the client's reachability at its apparent address, and 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.

### 2.3. Applications

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 in response to a request containing the Echo option. 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 state or time with a requesting client. A server MUST NOT synchronize state or

time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of 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 [I-D.ietf-core-object-security].
  - \* 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 expect by 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 2.2. 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. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client, addressing the issue described in Section 1.2. 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 blockwise request operations on a single resource. The requests must be integrity protected in order to protect against interchange of blocks between different message bodies.

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.

#### 3.1. Option Format

The Request-Tag option is not critical, is safe to forward, repeatable, and part of the cache key, see Figure 3, 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 3: Request-Tag Option Summary

[ Note to RFC editor: If this document is released before core-object-security, then the following paragraph and the "E"/"U" columns above need to move into core-object-security, as they are defined in that draft. ]

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 [I-D.ietf-core-object-security]): 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 blockwise 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.

### 3.2. 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 blockwise 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. They can still be treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when the later message carries Block1 number 0.

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.

### 3.3. Setting the Request-Tag

For each separate blockwise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. 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 MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.4.

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.

### 3.4. Applications

#### 3.4.1. 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 blockwise 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. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise 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.

#### 3.4.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise 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 blockwise 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 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.

#### 3.4.3. 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.hartke-core-stateless] Section 3.1.

#### 3.5. 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. Were it a single option, they would need to employ some length/value scheme to avoid confusing requests without a Request-Tag option with requests that carry a zero-length request tag.

In earlier versions of this draft, 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].

### 3.6. 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 3.4.1) 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.

### 4. Block2 / ETag Processing

The same security properties as in Section 3.4.1 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 3.4.1, 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.

### 5. Token Processing

As described in Section 1.3, the client must be able to verify that a response corresponds to a particular request. This section updates the Token processing in Section 5.3.1 of [RFC7252] by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the client **MUST NOT** reuse tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach **SHOULD** be followed.

### 6. 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.

An Echo value with 64 (pseudo-)random bits gives the same theoretical security level against forgeries as a 64-bit MAC (as used in e.g. AES\_128\_CCM\_8). In practice, forgery of an Echo option value is much harder as an attacker must also forge the MAC in the security protocol. 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 MUST 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.

Servers are not allowed to use wall clock time for timestamps, as wall clock time is not monotonic. Furthermore, an attacker may be able to affect the server's wall clock 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.

## 7. 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. The use of wall clock time is not allowed (see Section 6)

and there also privacy reasons, e.g. it may reveal 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.

## 8. 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 4: CoAP Option Numbers

## 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>>.
- [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>>.

### 9.2. Informative References

- [I-D.hartke-core-stateless]  
Hartke, K., "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", draft-hartke-core-stateless-01 (work in progress), September 2018.
- [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-15 (work in progress), August 2018.
- [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-03 (work in progress), October 2018.
- [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>>.

## 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 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 64-bit random values and 32-bit timestamps, the size of the Echo option value is 8 bytes and the amount of server state is  $12n$  bytes, where  $n$  is the number of active Echo Option values. 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. 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 7 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 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 3.4.1) 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 blockwise request operations happen rarely (in OSCORE, if there is always only one request blockwise 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 Major changes since draft-ietf-core-echo-request-tag-01:
  - \* Follow-up changes after the "relying on blockwise" 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 blockwise to treat it as part of the cache-key
  - \* Dropped use case about OSCORE outer-blockwise (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 Jim Schaad for providing valuable input to the draft.

#### Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com



John Mattsson  
Ericsson AB

Email: [john.mattsson@ericsson.com](mailto:john.mattsson@ericsson.com)

Goeran Selander  
Ericsson AB

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

CORE  
Internet-Draft  
Intended status: Standards Track  
Expires: August 30, 2019

M. Boucadair  
Orange  
T. Reddy  
McAfee  
J. Shallow  
NCC Group  
February 26, 2019

Constrained Application Protocol (CoAP) Hop Limit Option  
draft-ietf-core-hop-limit-03

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 August 30, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	2
3. Hop-Limit Option . . . . .	3
4. IANA Considerations . . . . .	4
4.1. CoAP Response Code . . . . .	4
4.2. CoAP Option Number . . . . .	5
5. Security Considerations . . . . .	5
6. Acknowledgements . . . . .	5
7. References . . . . .	5
7.1. Normative References . . . . .	5
7.2. Informative References . . . . .	6
Authors' Addresses . . . . .	6

## 1. Introduction

More and more applications are using Constrained Application Protocol (CoAP) [RFC7252] as a communication protocol between involved application agents. For example, [I-D.ietf-dots-signal-channel] specifies how CoAP is used as a distributed denial-of-service (DDoS) attack signaling protocol seeking for help from 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. To prevent such loops, this document defines a new CoAP option, called Hop-Limit (Section 3), which is inserted in particular by on-path proxies. Also, the document defines a new CoAP Response Code (Section 4.1) to report loops together with relevant diagnostic information to ease troubleshooting.

## 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 Hop-Limit option (see Section 4.2) is an elective option used to detect and prevent infinite loops when proxies are involved. The option is not repeatable. Therefore, any message carrying multiple Hop-Limit options MUST be rejected using 4.00 (Bad Request) error message.

The value of the Hop-Limit option is encoded as an 8-bit unsigned integer (see Section 3.2 of [RFC7252]). This value MUST be between 1 and 255 inclusive. CoAP messages 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 which 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 which does not understand the Hop-Limit option must follow the recommendations in Section 5.7.1 of [RFC7252] for caching. Note that loops which involve only such proxies won't be detected. Nevertheless, the presence of such proxies won't prevent infinite loop detection if at least one CoAP proxy which support the Hop-Limit option is involved in the loop.

A CoAP proxy which understands the Hop-Limit option MAY be instructed, using a configuration parameter, to insert a Hop-Limit option when relaying a request which do 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 to be 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. 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 messages (i.e., ignore the value of Hop-Limit received in a message). This modification should be done with caution in case proxy-forwarded traffic repeatedly crosses the administrative domain boundary in a loop and so Hop-Limit detection gets broken.

Otherwise, a CoAP proxy which understands the Hop-Limit option MUST decrement the value of the option by 1 prior to forwarding it. A

CoAP proxy which 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 less 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 less value of the Hop-Limit option is used instead.

CoAP messages MUST NOT be forwarded if the Hop-Limit option is set to '0' after decrement. Messages 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.

To ease debugging and troubleshooting, the CoAP proxy which detects a loop SHOULD include its information (e.g., proxy name, proxy alias, IP address) in the diagnostic payload under the conditions detailed in Section 5.5.2 of [RFC7252]. That information MUST NOT include any space character.

Each intermediate proxy involved in relaying a TBA1 (Hop Limit Reached) error message SHOULD prepend its own information in the diagnostic payload with a space character used as separator. Only one information per proxy SHOULD appear in the diagnostic payload. Doing so allows to limit the size of the TBA1 (Hop Limit Reached) error message, and to ease correlation with hops count.

#### 4. IANA Considerations

##### 4.1. 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 1: CoAP Response Codes

This document suggests 5.06 as a code to be assigned for the new response code.

Editorial Note: Please update TBA1 statements within the document with the assigned code.

#### 4.2. 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	C	U	N	R	Name	Reference
TBA2					Hop-Limit	[RFCXXXX]

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 2: CoAP Option Number

#### 5. Security Considerations

Security considerations related to CoAP proxying are discussed in Section 11.2 of [RFC7252].

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 which 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 response.

#### 6. 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, and Jim Schaad for the reviews.

#### 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>>.
- [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-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-28 (work in progress), January 2019.

## Authors' Addresses

Mohamed Boucadair  
Orange  
Rennes 35000  
France

Email: [mohamed.boucadair@orange.com](mailto:mohamed.boucadair@orange.com)

Tirumaleswar Reddy  
McAfee, Inc.  
Embassy Golf Link Business Park  
Bangalore, Karnataka 560071  
India

Email: [kondtir@gmail.com](mailto:kondtir@gmail.com)

Jon Shallow  
NCC Group  
United Kingdom

Email: [jon.shallow@nccgroup.com](mailto:jon.shallow@nccgroup.com)

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 12, 2019

Z. Shelby  
ARM  
M. Koster  
SmartThings  
C. Groves

J. Zhu  
Huawei  
B. Silverajan, Ed.  
Tampere University  
March 11, 2019

Reusable Interface Definitions for Constrained RESTful Environments  
draft-ietf-core-interfaces-14

Abstract

This document defines a set of Constrained RESTful Environments (CoRE) Link Format Interface Descriptions [RFC6690] applicable for use in constrained environments. These include the: Actuator, Parameter, Read-only parameter, Sensor, Batch, Linked Batch and Link List interfaces.

The Batch, Linked Batch and Link List interfaces make use of resource collections. This document further describes how collections relate to interfaces.

Many applications require a set of interface descriptions in order provide the required functionality. This document defines an Interface Description attribute value to describe resources conforming to a particular interface.

Editor's notes:

- o The git repository for the draft is found at <https://github.com/core-wg/interfaces>

Status of This Memo

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

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



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

This Internet-Draft will expire on September 12, 2019.

#### Copyright Notice

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

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

#### Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Collections . . . . .	4
3.1. Introduction to Collections . . . . .	5
3.2. Use Cases for Collections . . . . .	5
3.3. Collection Types . . . . .	6
3.4. Content-Formats for Collections . . . . .	6
3.5. Link Embedding . . . . .	7
3.6. Links and Items in Collections . . . . .	7
3.7. Queries on Collections . . . . .	8
3.8. Observing Collections . . . . .	8
4. Interface Descriptions . . . . .	9
4.1. Link List . . . . .	11
4.2. Batch . . . . .	11
4.3. Linked Batch . . . . .	12
4.4. Sensor . . . . .	13
4.5. Parameter . . . . .	14
4.6. Read-only Parameter . . . . .	14
4.7. Actuator . . . . .	14
5. Security Considerations . . . . .	15
6. IANA Considerations . . . . .	15
6.1. Link List . . . . .	15
6.2. Batch . . . . .	16
6.3. Linked Batch . . . . .	16

6.4. Sensor . . . . .	16
6.5. Parameter . . . . .	17
6.6. Read-only parameter . . . . .	17
6.7. Actuator . . . . .	17
7. Acknowledgements . . . . .	17
8. Contributors . . . . .	18
9. Changelog . . . . .	18
10. References . . . . .	22
10.1. Normative References . . . . .	22
10.2. Informative References . . . . .	22
Appendix A. Current Usage of Interfaces . . . . .	23
A.1. Constrained RESTful Environments (CoRE) Link Format (IETF) . . . . .	23
A.2. Open Connectivity Foundation (OCF) . . . . .	24
Authors' Addresses . . . . .	24

## 1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC8288] in constrained environments. SenML [RFC8428] is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

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. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained origin servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for interface description ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format interface descriptions for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction.

An interface description describes a resource in terms of its associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, and actuators.

A set of collection types is defined for organizing resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document first defines the concept of collection interface descriptions. It then defines a number of generic interface descriptions that may be used in constrained environments. Several of these interface descriptions utilise collections.

Whilst this document assumes the use of CoAP [RFC7252], the REST interfaces described can also be realized using 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].

This document requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This document makes use of the following additional terminology:

**Gradual Reveal:** A REST design where resources are discovered progressively using Web Linking.

**Interface Description:** The Interface Description describes the generic REST interface to interact with a resource or a set of resources. Its use is described via the Interface Description 'if' attribute which is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource.

**Resource Discovery:** The process allowing a client to identify resources being hosted on an origin server.

## 3. Collections

### 3.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. [RFC6573] describes the "item" and "collection" Link Relation. An "item" link relation identifies a member of collection. A "collection" indicates the collection that an item is a member of. For example, a collection might be a resource representing a catalog of products, while an item is a resource related to an individual product.

Section 1.2.2/[RFC6690] also describes resource collections.

This document uses the concept of "collection" and applies it to interface descriptions. A collection interface description consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection interface descriptions described in this document are Link List, Batch and Linked Batch.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by SenML, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

### 3.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update from a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

### 3.3. Collection Types

There are three collection types defined in this document:

Collection Type	if=
Link List	core.ll
Batch	core.b
Linked Batch	core.lb

Table 1: Collection Type Summary

The interface description defined in this document offer a deeper explanation of the methods that may be applied to the three collections.

### 3.4. Content-Formats for Collections

The collection interfaces can use the CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the Accept header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

### 3.5. Link Embedding

Collections may provide resource encapsulation by supporting link embedding. Link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. This is analogous to an image tag (link) causing the image to display inline in a browser window. Link embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. Performing a GET on a collection resource may return a single representation containing all of the embedded linked resources. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single SenML data object.

A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document.

### 3.6. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes links to resources with absolute paths as well as links that point to other network locations, if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC8288]. Links to other collections are formed per [RFC3986].

Examples of links:

</sen/>;if="core.lb": Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/temp>;rt="temperature": A link to the temp resource with an absolute path.

<temp>;rt="temperature": Link to the temp subresource of the collection in which this link appears.

<temp>;anchor="/sen/": A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears, but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in [RFC7396].

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

### 3.7. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

### 3.8. Observing Collections

Resource Observation via [I-D.ietf-core-dynlink] using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection's resource returns the collection representation. Observation Responses, or notifications, SHOULD provide the collection representations in SenML Content-Format. Notifications MAY include multiple observations of the collection resource, with SenML time stamps indicating the observation times.

#### 4. Interface Descriptions

This section defines REST interfaces for Sensor, Parameter, Read-Only Parameter and Actuator resource types, in addition to the Link List, Batch and Linked Batch collection types. Each type is described along with its Interface Description attribute value, valid methods and content formats. These are shown for each interface in the table below.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this document. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.



Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	senml, text/plain
Parameter	core.p	GET, PUT	senml, text/plain
Read-only Parameter	core.rp	GET	senml, text/plain
Actuator	core.a	GET, PUT, POST	senml, text/plain

Table 2: Interface Description Summary

The following is an example of links in the CoRE Link Format using these interface descriptions.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.lt";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb"

```

Figure 1: Binding Interface Example

#### 4.1. Link List

Link List is the base interface to provide gradual reveal of resources on a CoRE origin server. It is used to retrieve (GET) a list of resources on an origin server. The GET request SHOULD contain an Accept option with the application/link-format content format. However if the resource does not support any other form of content-format the Accept option MAY be elided.

Note: The use of an Accept option with application/link-format is recommended even though it is not strictly needed for the Link List interface because this interface is extended by the batch and linked batch interfaces where different content-formats are possible.

The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on an origin server.

The following example interacts with a Link List /d/ containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d/ (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

#### 4.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface description supports the same methods as its sub-resources, and can be used to read (GET), update (PUT) or apply (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous. Hence, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

The following example interacts with a Batch /s/ with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s/  
Res: 2.05 Content (application/senml+json)  
[  
  { "bn": "example.com/s/" },  
  { "n": "light", "v": 123, "u": "lx" },  
  { "n": "temp", "v": 27.2, "u": "Cel" },  
  { "n": "humidity", "v": 80, "u": "%RH" }  
]
```

#### 4.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC8288] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the origin server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /1/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" }
]
```

```
Req: POST /1/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /1/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }
]
```

```
Req: DELETE /1/
Res: 2.02 Deleted
```

#### 4.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/s/" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

#### 4.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and updating a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5
```

```
Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

#### 4.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

#### 4.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated

(PUT). In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

The following example shows requests for reading, setting and toggling an actuator (turning on a LED).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

```
Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed
```

```
Req: POST /a/1/led (text/plain)
Res: 2.04 Changed
```

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

## 5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service. Conversely, a malicious client could attempt to write to arbitrary resources on a poorly implemented server described in a linked batch.

## 6. IANA Considerations

This document registers the following CoRE Interface Description (if=) Link Target Attribute Values.

### 6.1. Link List

Attribute Value: core.ll

Description: The Link List interface is used to retrieve a list of resources on an origin server.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.2. Batch

Attribute Value: core.b

Description: The Batch interface is used to manipulate a collection of sub-resources at the same time.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.3. Linked Batch

Attribute Value: core.lb

Description: The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.4. Sensor

Attribute Value: core.s

Description: The Sensor interface allows the value of a sensor resource to be read.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.5. Parameter

Attribute Value: core.p

Description: The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read or update.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.6. Read-only parameter

Attribute Value: core.rp

Description: The Read-only Parameter interface allows configuration parameters to be read but not updated.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 6.7. Actuator

Attribute Value: core.a

Description: The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read or the actuator value can be updated. In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

## 7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further



requirements for interface descriptions have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document. Ari Keraenen provided updated SenML examples. Christian Amsuss supplied a comprehensive review of draft -12.

## 8. Contributors

Matthieu Vial  
Schneider-Electric  
Grenoble  
France

Phone: +33 (0)47657 6522  
EMail: matthieu.vial@schneider-electric.com

## 9. Changelog

Changes from -13 to -14:

- o Version update, with changes in editor's contact information

Changes from -12 to -13:

- o SenML examples now use the Base Name (bn) labels from RFC 8428
- o Security considerations discusses client misuse of linked batches

Changes from -11 to -12:

- o Removed all text referring to function sets/profiles
- o Clarified list collections
- o Content-formats for collections and items rectified
- o Simplified Appendix A and removed Appendix B

Changes from -10 to -11:

- o Added a new Section 3.4 for Link Embedding
- o Updated examples in Section 3.5
- o Removed "Service Discovery" from Terminologies
- o Removed discussion of function sets

Changes from -09 to -10:

- o Section 1: Amendments to remove discussing properties. \*
- o New author and editor added.

Changes from -08 to -09:

- o Section 3.6: Modified to indicate that the entire collection resource is returned.
- o General: Added editor's note with open issues.

Changes from -07 to -08:

- o Section 3.3: Modified Accepts to Accept header option.
- o Addressed the editor's note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to -07:

- o Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- o Addressed the editor's note in Section 4.2.
- o Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- o Updated the abstract.
- o Section 1: Updated introduction.
- o Section 2: Alphabetised the order
- o Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- o Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- o General: Changed interface type to interface description as that is the term defined in RFC6690.
- o Removed section on future interfaces.

- o Section 8: Updated IANA considerations.
- o Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- o Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- o Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- o The WADL description has been removed.
- o Fixed minor typos.
- o Updated references.

Changes from -03 to -04:

- o Fixed tickets #385 and #386.
- o Changed abstract and intro to better describe content.
- o Focus on Interface and not function set/profiles in intro.
- o Changed references from draft-core-observe to RFC7641.
- o Moved Function sets and Profiles to section after Interfaces.
- o Moved Observe Attributes to the Link Binding section.
- o Add a Collection section to describe the collection types.
- o Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- o Added lt and gt to binding format section.
- o Added pmin and pmax observe parameters to Observation Attributes.
- o Changed the definition of lt and gt to limit crossing.
- o Added definitions for getattr and setattr to WADL.
- o Added getattr and setattr to observable interfaces.

- o Removed query parameters from Observe definition.
- o Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02:

- o Updated the date and version, fixed references.
- o "Removed pmin and pmax observe parameters "[Ticket #336]"."

Changes from -00 to WG Document -01

- o Improvements to the Function Set section.

Changes from -05 to WG Document -00

- o Updated the date and version.

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.
- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

## 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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

### 10.2. Informative References

- [I-D.ietf-core-dynlink]  
Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-08 (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-19 (work in progress), January 2019.
- [OIC-Core]  
"OIC Resource Type Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OIC-SmartHome]  
"OIC Smart Home Device Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OMA-TS-LWM2M]  
"Lightweight Machine to Machine Technical Specification", 2016, <<http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>>.
- [oneM2MTS0008]  
"TS 0008 v1.3.2 CoAP Protocol Binding", 2016, <<http://www.onem2m.org/technical/published-documents>>.

- [oneM2MTS0023] "TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2016,  
<<http://www.onem2m.org/technical/published-documents>>.
- [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>>.
- [RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012,  
<<https://www.rfc-editor.org/info/rfc6573>>.
- [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>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014,  
<<https://www.rfc-editor.org/info/rfc7396>>.
- [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>>.

## Appendix A. Current Usage of Interfaces

Editor's note: This appendix will be removed. It is only included for information.

This appendix analyses the current landscape with regards the definition and use of collections and interfaces. This should be considered when considering the scope of this document.

### A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)

[RFC6690] assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful. It highlights that collections are often used for these interfaces.

Whilst 3.2/[RFC6690] defines a new Interface Description 'if' attribute the procedures around it are about the naming of the interface not what information should be included in the documentation about the interface.

#### A.2. Open Connectivity Foundation (OCF)

The OIC Core Specification [OIC-Core] most closely aligns with the work in this specification. It makes use of interface descriptions as per [RFC6690] and has registered several interface identifiers (<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#if-link-target-att-value>). These interface descriptors are similar to those defined in this specification. From a high level perspective:

```
links list:   OCF (oic.if.ll) -> IETF (core.ll)
               Note: it's called "link list" in the IETF.
linked batch: OCF (oic.if.b) -> IETF (core.lb)
read-only:    OCF (oic.if.r) -> IETF (core.rp)
read-write:   OCF (oic.if.rw) -> IETF (core.p)
actuator:     OCF (oic.if.a) -> IETF (core.a)
sensor:       OCF (oic.if.s) -> IETF (core.s)
batch:        No OCF equivalent -> IETF (core.b)
```

Some of the OCF interfaces make use of collections.

The OIC Core specification does not use the concept of function sets. It does however discuss the concept of profiles. The OCF defines two sets of documents. The core specification documents such as [OIC-Core] and vertical profile specification documents which provide specific information for specific applications. The OIC Smart Home Device Specification [OIC-SmartHome] is one such specification. It provides information on the resource model, discovery and data types.

#### Authors' Addresses

Zach Shelby  
ARM  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach.shelby@arm.com

Michael Koster  
SmartThings  
665 Clyde Avenue  
Mountain View 94043  
USA

Email: michael.koster@smarththings.com

Christian Groves  
Australia

Email: cngroves.std@gmail.com

Jintao Zhu  
Huawei  
No.127 Jinye Road, Huawei Base, High-Tech Development District  
Xi'an, Shaanxi Province  
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)  
Tampere University  
Kalevantie 4  
Tampere FI-33100  
Finland

Email: bilhanan.silverajan@tuni.fi



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: September 9, 2019

T. Fossati  
ARM  
K. Hartke  
Ericsson  
C. Bormann  
Universitaet Bremen TZI  
March 08, 2019

Multipart Content-Format for CoAP  
draft-ietf-core-multipart-ct-03

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 September 9, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Requirements Language . . . . .	3
2. Multipart Content-Format Encoding . . . . .	3
3. Usage Examples . . . . .	4
3.1. Observing Resources . . . . .	4
4. Implementation hints . . . . .	4
5. IANA Considerations . . . . .	5
5.1. Registration of media type application/multipart-core . .	5
5.2. Registration of a Content-Format identifier for application/multipart-core . . . . .	7
6. Security Considerations . . . . .	7
7. References . . . . .	7
7.1. Normative References . . . . .	7
7.2. Informative References . . . . .	7
Acknowledgements . . . . .	8
Authors' Addresses . . . . .	8

## 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 into a single message, such as a CoAP [RFC7252] request or response body, with minimal framing overhead, each along with a CoAP Content-Format identifier.

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.

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 different formats to be played out in that sequence.

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 allows to indicate that an optional part is not present by substituting a null value for the representation of the part.

A third situation that is common only ever has a single representation in the sequence, which is one of a set of formats possible. This kind of union 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 application/multipart-core.

### 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. The second, fourth, sixth, etc. element is a byte string containing a representation, or the value "null" if an optional part is indicated as not given. The first, third, fifth, etc. element 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 [I-D.ietf-cbor-cddl] 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 Examples

#### 3.1. Observing Resources

When a client registers to observe a resource [RFC7641] 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. The possible resulting sequence of notifications is shown in Figure 1.

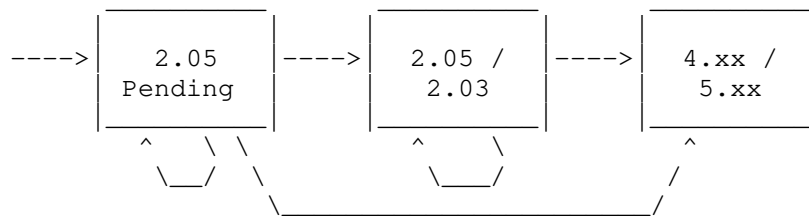


Figure 2: Sequence of Notifications:

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.

## 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-10 (work in progress), March 2019.
- [I-D.ietf-cbor-cddl]  
Birkholz, H., Vigano, C., and C. Bormann, "Concise data  
definition language (CDDL): a notational convention to  
express CBOR and JSON data structures", draft-ietf-cbor-  
cddl-07 (work in progress), February 2019.
- [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>>.

#### 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](mailto:thomas.fossati@arm.com)

Klaus Hartke  
Ericsson  
Torshamnsgatan 23  
Stockholm SE-16483  
Sweden

Email: [klaus.hartke@ericsson.com](mailto: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  
Updates: 7252 (if approved)  
Intended status: Standards Track  
Expires: September 7, 2019

G. Selander  
J. Mattsson  
F. Palombini  
Ericsson AB  
L. Seitz  
RISE SICS  
March 06, 2019

Object Security for Constrained RESTful Environments (OSCORE)  
draft-ietf-core-object-security-16

Abstract

This document defines Object Security for Constrained RESTful Environments (OSCORE), a method for application-layer protection of the Constrained Application Protocol (CoAP), using CBOR Object Signing and Encryption (COSE). OSCORE provides end-to-end protection between endpoints communicating using CoAP or CoAP-mappable HTTP. OSCORE is designed for constrained nodes and networks supporting a range of proxy operations, including translation between different transport protocols.

Although being an optional functionality of CoAP, OSCORE alters CoAP options processing and IANA registration. Therefore, this document updates [RFC7252].

Status of This Memo

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

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

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

This Internet-Draft will expire on September 7, 2019.

## Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	4
1.1. Terminology . . . . .	6
2. The OSCORE Option . . . . .	7
3. The Security Context . . . . .	7
3.1. Security Context Definition . . . . .	8
3.2. Establishment of Security Context Parameters . . . . .	10
3.3. Requirements on the Security Context Parameters . . . . .	12
4. Protected Message Fields . . . . .	13
4.1. CoAP Options . . . . .	14
4.2. CoAP Header Fields and Payload . . . . .	23
4.3. Signaling Messages . . . . .	23
5. The COSE Object . . . . .	24
5.1. ID Context and 'kid context' . . . . .	25
5.2. AEAD Nonce . . . . .	26
5.3. Plaintext . . . . .	27
5.4. Additional Authenticated Data . . . . .	28
6. OSCORE Header Compression . . . . .	29
6.1. Encoding of the OSCORE Option Value . . . . .	30
6.2. Encoding of the OSCORE Payload . . . . .	31
6.3. Examples of Compressed COSE Objects . . . . .	31
7. Message Binding, Sequence Numbers, Freshness, and Replay Protection . . . . .	34
7.1. Message Binding . . . . .	34
7.2. Sequence Numbers . . . . .	34
7.3. Freshness . . . . .	34
7.4. Replay Protection . . . . .	35
7.5. Losing Part of the Context State . . . . .	36
8. Processing . . . . .	37
8.1. Protecting the Request . . . . .	37
8.2. Verifying the Request . . . . .	37
8.3. Protecting the Response . . . . .	39

8.4. Verifying the Response . . . . .	40
9. Web Linking . . . . .	42
10. CoAP-to-CoAP Forwarding Proxy . . . . .	42
11. HTTP Operations . . . . .	43
11.1. The HTTP OSCORE Header Field . . . . .	43
11.2. CoAP-to-HTTP Mapping . . . . .	44
11.3. HTTP-to-CoAP Mapping . . . . .	45
11.4. HTTP Endpoints . . . . .	45
11.5. Example: HTTP Client and CoAP Server . . . . .	46
11.6. Example: CoAP Client and HTTP Server . . . . .	47
12. Security Considerations . . . . .	48
12.1. End-to-end Protection . . . . .	48
12.2. Security Context Establishment . . . . .	49
12.3. Master Secret . . . . .	49
12.4. Replay Protection . . . . .	50
12.5. Client Aliveness . . . . .	50
12.6. Cryptographic Considerations . . . . .	50
12.7. Message Segmentation . . . . .	51
12.8. Privacy Considerations . . . . .	51
13. IANA Considerations . . . . .	52
13.1. COSE Header Parameters Registry . . . . .	52
13.2. CoAP Option Numbers Registry . . . . .	53
13.3. CoAP Signaling Option Numbers Registry . . . . .	54
13.4. Header Field Registrations . . . . .	54
13.5. Media Type Registrations . . . . .	54
13.6. CoAP Content-Formats Registry . . . . .	56
13.7. OSCORE Flag Bits Registry . . . . .	56
13.8. Expert Review Instructions . . . . .	57
14. References . . . . .	58
14.1. Normative References . . . . .	58
14.2. Informative References . . . . .	59
Appendix A. Scenario Examples . . . . .	62
A.1. Secure Access to Sensor . . . . .	62
A.2. Secure Subscribe to Sensor . . . . .	63
Appendix B. Deployment Examples . . . . .	64
B.1. Security Context Derived Once . . . . .	64
B.2. Security Context Derived Multiple Times . . . . .	66
Appendix C. Test Vectors . . . . .	71
C.1. Test Vector 1: Key Derivation with Master Salt . . . . .	72
C.2. Test Vector 2: Key Derivation without Master Salt . . . . .	73
C.3. Test Vector 3: Key Derivation with ID Context . . . . .	75
C.4. Test Vector 4: OSCORE Request, Client . . . . .	76
C.5. Test Vector 5: OSCORE Request, Client . . . . .	77
C.6. Test Vector 6: OSCORE Request, Client . . . . .	79
C.7. Test Vector 7: OSCORE Response, Server . . . . .	80
C.8. Test Vector 8: OSCORE Response with Partial IV, Server . . . . .	81
Appendix D. Overview of Security Properties . . . . .	82
D.1. Threat Model . . . . .	82

D.2. Supporting Proxy Operations . . . . .	83
D.3. Protected Message Fields . . . . .	84
D.4. Uniqueness of (key, nonce) . . . . .	85
D.5. Unprotected Message Fields . . . . .	86
Appendix E. CDDL Summary . . . . .	89
Acknowledgments . . . . .	90
Authors' Addresses . . . . .	90

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol, designed for constrained nodes and networks [RFC7228], and may be mapped from HTTP [RFC8075]. CoAP specifies the use of proxies for scalability and efficiency and references DTLS [RFC6347] for security. CoAP-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require DTLS or TLS [RFC8446] to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of, the message payload and metadata in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports the Observe [RFC7641], Block-wise [RFC7959], and No-Response [RFC7967] options, as well as the PATCH and FETCH methods [RFC8132]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE essentially protects the RESTful interactions; the request method, the requested resource, the message payload, etc. (see Section 4). OSCORE protects neither the CoAP Messaging Layer nor the CoAP Token which may change between the endpoints, and those are therefore processed as defined in [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [RFC8323] differ only in terms of CoAP Messaging Layer, OSCORE can be applied to both unreliable and reliable transports (see Figure 1).

OSCORE works in very constrained nodes and networks, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. Examples of the use of OSCORE are given in Appendix A. OSCORE may be used over any underlying layer, such as e.g. UDP or TCP, and with non-IP transports (e.g.,

[I-D.bormann-6lo-coap-802-15-ie]). OSCORE may also be used in different ways with HTTP. OSCORE messages may be transported in HTTP, and OSCORE may also be used to protect CoAP-mappable HTTP messages, as described below.

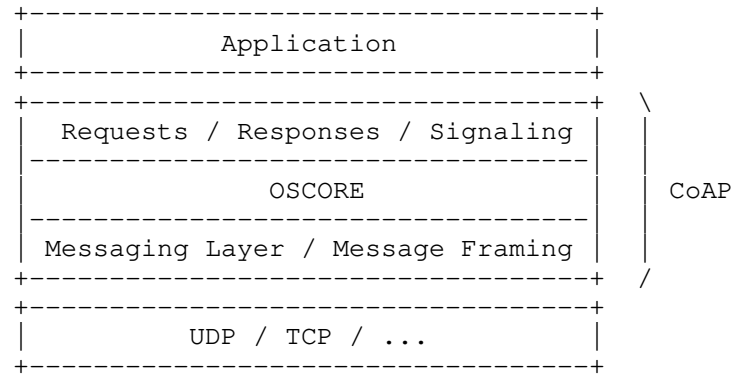


Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE is designed to protect as much information as possible while still allowing CoAP proxy operations (Section 10). It works with existing CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used with OSCORE, as specified in Section 11. OSCORE may be used together with TLS or DTLS over one or more hops in the end-to-end path, e.g. transported with HTTPS in one hop and with plain CoAP in another hop. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-band or with a key establishment protocol (see Section 3.2). The technical solution builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled in CoAP with a new option (Section 2), and in HTTP with a new header field (Section 11.1) and content type (Section 13.5). The solution transforms a CoAP/HTTP message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is a CoAP/HTTP message related to the original message in the following way: the original CoAP/HTTP message is translated to CoAP (if not already in CoAP) and protected in a COSE object. The encrypted message fields of this COSE object are transported in the CoAP payload/HTTP body of the OSCORE message, and the OSCORE option/

header field is included in the message. A sketch of an exchange of OSCORE messages, in the case of the original message being CoAP, is provided in Figure 2. The use of OSCORE with HTTP is detailed in Section 11.

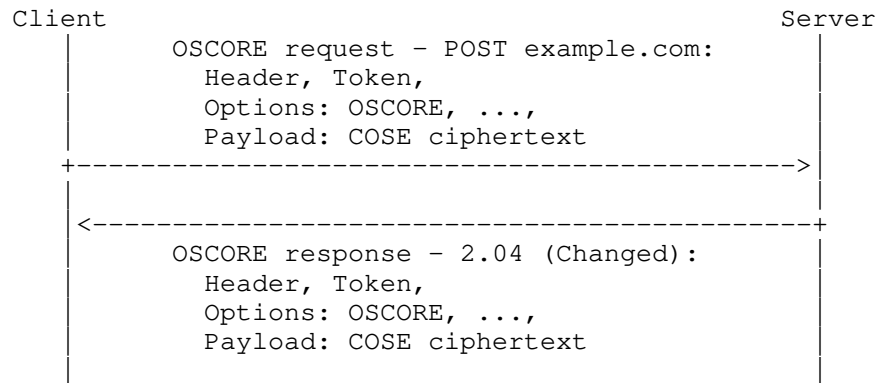


Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts.

### 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], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, and constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end path. The concept "hop-by-hop" (as in "hop-by-hop encryption" or "hop-by-hop fragmentation") opposed to "end-to-end", is used in this document to indicate that the messages are processed accordingly in the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote that the message is not passed up to the CoAP Request/Response Layer (see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key, Recipient ID/Key, ID Context, and Common IV are defined in Section 3.1.

## 2. The OSCORE Option

The OSCORE option defined in this section (see Figure 3, which extends Table 4: Options of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Sections 5 and 6). The OSCORE option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1	x				OSCORE	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable  
 (\*) See below.

Figure 3: The OSCORE Option

The OSCORE option includes the OSCORE flag bits (Section 6), the Sender Sequence Number, the Sender ID, and the ID Context when these fields are present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits are all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an OSCORE option SHALL treat it as malformed and reject it.

A successful response to a request with the OSCORE option SHALL contain the OSCORE option. Whether error responses contain the OSCORE option depends on the error type (see Section 8).

For CoAP proxy operations, see Section 10.

## 3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD, [RFC5116]) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a shared secret and a key derivation function.



### 3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCORE. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients and servers need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus, the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 4.

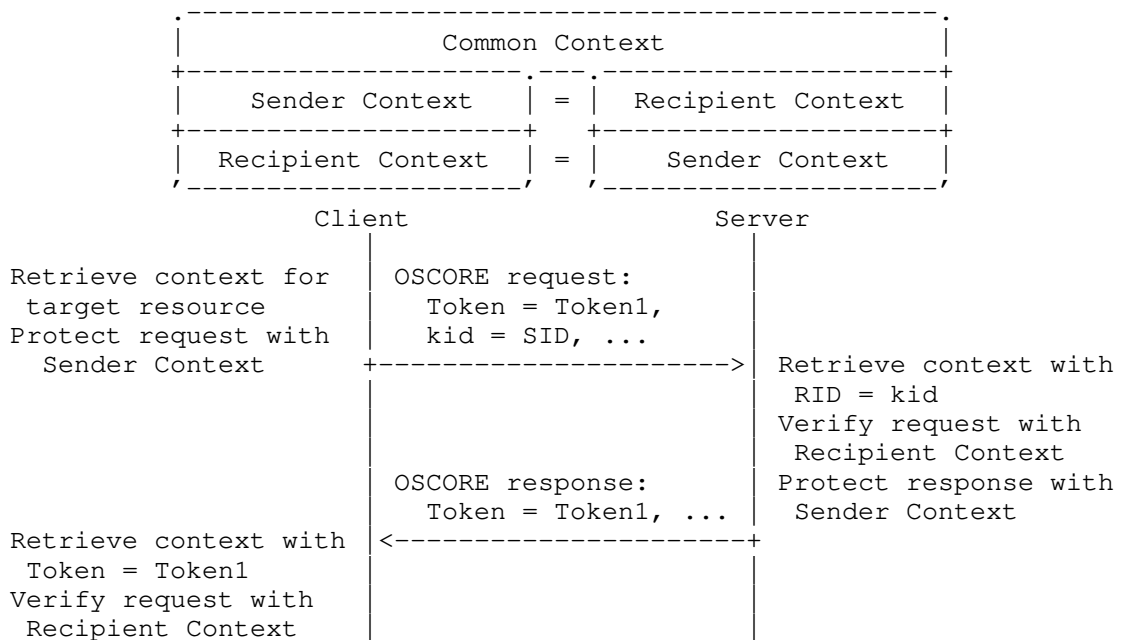


Figure 4: Retrieval and Use of the Security Context

The Common Context contains the following parameters:

- o AEAD Algorithm. The COSE AEAD algorithm to use for encryption.
- o HKDF Algorithm. An HMAC-based key derivation function HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.
- o Master Secret. Variable length, random byte string (see Section 12.3) used to derive AEAD keys and Common IV.
- o Master Salt. Optional variable length byte string containing the salt used to derive AEAD keys and Common IV.
- o ID Context. Optional variable length byte string providing additional information to identify the Common Context and to derive AEAD keys and Common IV. The use of ID Context is described in Section 5.1.
- o Common IV. Byte string derived from Master Secret, Master Salt, and ID Context. Used to generate the AEAD Nonce (see Section 5.2). Same length as the nonce of the AEAD Algorithm.

The Sender Context contains the following parameters:

- o Sender ID. Byte string used to identify the Sender Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Sender Key. Byte string containing the symmetric AEAD key to protect messages to send. Derived from Common Context and Sender ID. Length is determined by the AEAD Algorithm.
- o Sender Sequence Number. Non-negative integer used by the sender to enumerate requests and certain responses, e.g. Observe notifications. Used as 'Partial IV' [RFC8152] to generate unique AEAD nonces. Maximum value is determined by the AEAD Algorithm. Initialization is described in Section 3.2.2.

The Recipient Context contains the following parameters:

- o Recipient ID. Byte string used to identify the Recipient Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Recipient Key. Byte string containing the symmetric AEAD key to verify messages received. Derived from Common Context and Recipient ID. Length is determined by the AEAD Algorithm.

- o Replay Window (Server only). The replay window to verify requests received. Replay protection is described in Section 7.4 and Section 3.2.2.

All parameters except Sender Sequence Number and Replay Window are immutable once the security context is established. An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them when needed. Alternatively, an endpoint may free up memory by not storing the Master Secret and Master Salt after the other parameters have been derived.

Endpoints MAY operate as both client and server and use the same security context for those roles. Independent of being client or server, the endpoint protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The endpoints MUST NOT change the Sender/Recipient ID when changing roles. In other words, changing the roles does not change the set of AEAD keys to be used.

### 3.2. Establishment of Security Context Parameters

Each endpoint derives the parameters in the security context from a small set of input parameters. The following input parameters SHALL be pre-established:

- o Master Secret
- o Sender ID
- o Recipient ID

The following input parameters MAY be pre-established. In case any of these parameters is not pre-established, the default value indicated below is used:

- o AEAD Algorithm
  - \* Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- o Master Salt
  - \* Default is the empty byte string
- o HKDF Algorithm
  - \* Default is HKDF SHA-256
- o Replay Window

- \* Default is DTLS-type replay protection with a window size of 32 [RFC6347]

All input parameters need to be known to and agreed on by both endpoints, but the replay window may be different in the two endpoints. The way the input parameters are pre-established, is application specific. Considerations of security context establishment are given in Section 12.2 and examples of deploying OSCORE in Appendix B.

### 3.2.1. Derivation of Sender Key, Recipient Key, and Common IV

The HKDF MUST be one of the HMAC-based HKDF [RFC5869] algorithms defined for COSE [RFC8152]. HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key, Recipient Key, and Common IV SHALL be derived from the input parameters using the HKDF, which consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]:

```
output parameter = HKDF(salt, IKM, info, L)
```

where:

- o salt is the Master Salt as defined above
- o IKM is the Master Secret as defined above
- o info is the serialization of a CBOR array consisting of (the notation follows Appendix E):

```
info = [  
  id : bstr,  
  id_context : bstr / nil,  
  alg_aead : int / tstr,  
  type : tstr,  
  L : uint,  
]
```

where:

- o id is the Sender ID or Recipient ID when deriving Sender Key and Recipient Key, respectively, and the empty byte string when deriving the Common IV.
- o id\_context is the ID Context, or nil if ID Context is not provided.
- o alg\_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

- o type is "Key" or "IV". The label is an ASCII string, and does not include a trailing NUL byte.
- o L is the size of the key/nonce for the AEAD algorithm used, in bytes.

For example, if the algorithm AES-CCM-16-64-128 (see Section 10.2 in [RFC8152]) is used, the integer value for alg\_aead is 10, the value for L is 16 for keys and 13 for the Common IV. Assuming use of the default algorithms HKDF SHA-256 and AES-CCM-16-64-128, the extract phase of HKDF produces a pseudorandom key (PRK) as follows:

PRK = HMAC-SHA-256(Master Salt, Master Secret)

and as L is smaller than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation, and the Sender Key, Recipient Key, and Common IV are therefore the first 16 or 13 bytes of

output parameter = HMAC-SHA-256(PRK, info || 0x01)

where different info are used for each derived parameter and where || denotes byte string concatenation.

Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros. For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string. OSCORE sets the salt default value to empty byte string, which is converted to a string of zeroes (see Section 2.2 of [RFC5869]).

### 3.2.2. Initial Sequence Numbers and Replay Window

The Sender Sequence Number is initialized to 0.

The supported types of replay protection and replay window length is application specific and depends on how OSCORE is transported, see Section 7.4. The default is DTLS-type replay protection with a window size of 32 initiated as described in Section 4.1.2.6 of [RFC6347].

### 3.3. Requirements on the Security Context Parameters

To ensure unique Sender Keys, the quartet (Master Secret, Master Salt, ID Context, Sender ID) MUST be unique, i.e. the pair (ID Context, Sender ID) SHALL be unique in the set of all security contexts using the same Master Secret and Master Salt. This means that Sender ID SHALL be unique in the set of all security contexts

using the same Master Secret, Master Salt, and ID Context; such a requirement guarantees unique (key, nonce) pairs for the AEAD.

Different methods can be used to assign Sender IDs: a protocol that allows the parties to negotiate locally unique identifiers, a trusted third party (e.g., [I-D.ietf-ace-oauth-authz]), or the identifiers can be assigned out-of-band. The Sender IDs can be very short (note that the empty string is a legitimate value). The maximum length of Sender ID in bytes equals the length of AEAD nonce minus 6, see Section 5.2. For AES-CCM-16-64-128 the maximum length of Sender ID is 7 bytes.

To simplify retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint. If an endpoint has the same Recipient ID with different Recipient Contexts, i.e. the Recipient Contexts are derived from different Common Contexts, then the endpoint may need to try multiple times before verifying the right security context associated to the Recipient ID.

The ID Context is used to distinguish between security contexts. The methods used for assigning Sender ID can also be used for assigning the ID Context. Additionally, the ID Context can be used to introduce randomness into new Sender and Recipient Contexts (see Appendix B.2). ID Context can be arbitrarily long.

#### 4. Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from an HTTP message) into an OSCORE message, and vice versa. OSCORE protects as much of the original message as possible while still allowing certain proxy operations (see Sections 10 and 11). This section defines how OSCORE protects the message fields and transfers them end-to-end between client and server (in any direction).

The remainder of this section and later sections focus on the behavior in terms of CoAP messages. If HTTP is used for a particular hop in the end-to-end path, then this section applies to the conceptual CoAP message that is mappable to/from the original HTTP message as discussed in Section 11. That is, an HTTP message is conceptually transformed to a CoAP message and then to an OSCORE message, and similarly in the reverse direction. An actual implementation might translate directly from HTTP to OSCORE without the intervening CoAP representation.

Protection of Signaling messages (Section 5 of [RFC8323]) is specified in Section 4.3. The other parts of this section target Request/Response messages.

Message fields of the CoAP message may be protected end-to-end between CoAP client and CoAP server in different ways:

- o Class E: encrypted and integrity protected,
- o Class I: integrity protected only, or
- o Class U: unprotected.

The sending endpoint SHALL transfer Class E message fields in the ciphertext of the COSE object in the OSCORE message. The sending endpoint SHALL include Class I message fields in the Additional Authenticated Data (AAD) of the AEAD algorithm, allowing the receiving endpoint to detect if the value has changed in transfer. Class U message fields SHALL NOT be protected in transfer. Class I and Class U message field values are transferred in the header or options part of the OSCORE message, which is visible to proxies.

Message fields not visible to proxies, i.e., transported in the ciphertext of the COSE object, are called "Inner" (Class E). Message fields transferred in the header or options part of the OSCORE message, which is visible to proxies, are called "Outer" (Class I or U). There are currently no Class I options defined.

An OSCORE message may contain both an Inner and an Outer instance of a certain CoAP message field. Inner message fields are intended for the receiving endpoint, whereas Outer message fields are used to enable proxy operations.

#### 4.1. CoAP Options

A summary of how options are protected is shown in Figure 5. Note that some options may have both Inner and Outer message fields which are protected accordingly. Certain options require special processing as is described in Section 4.1.3.

Options that are unknown or for which OSCORE processing is not defined SHALL be processed as class E (and no special processing). Specifications of new CoAP options SHOULD define how they are processed with OSCORE. A new COAP option SHOULD be of class E unless it requires proxy processing. If a new CoAP option is of class U, the potential issues with the option being unprotected SHOULD be documented (see Appendix D.5).

## 4.1.1.1. Inner Options

Inner option message fields (class E) are used to communicate directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields present in the original CoAP message into the plaintext of the COSE object (Section 5.3), and then remove the Inner option message fields from the OSCORE message.

The processing of Inner option message fields by the receiving endpoint is specified in Sections 8.2 and 8.4.

No.	Name	E	U
1	If-Match	x	
3	Uri-Host		x
4	ETag	x	
5	If-None-Match	x	
6	Observe	x	x
7	Uri-Port		x
8	Location-Path	x	
TBD1	OSCORE		x
11	Uri-Path	x	
12	Content-Format	x	
14	Max-Age	x	x
15	Uri-Query	x	
17	Accept	x	
20	Location-Query	x	
23	Block2	x	x
27	Block1	x	x
28	Size2	x	x
35	Proxy-Uri		x
39	Proxy-Scheme		x
60	Size1	x	x
258	No-Response	x	x

E = Encrypt and Integrity Protect (Inner)  
 U = Unprotected (Outer)

Figure 5: Protection of CoAP Options



#### 4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations, see Appendix D.2.

The sending endpoint SHALL include the Outer option message field present in the original message in the options part of the OSCORE message. All Outer option message fields, including the OSCORE option, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Outer option message field.

The processing of Outer options by the receiving endpoint is specified in Sections 8.2 and 8.4.

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.4. Specifications that introduce repeatable Class I options MUST specify that proxies MUST NOT change the order of the instances of such an option in the CoAP message.

Note: There are currently no Class I option message fields defined.

#### 4.1.3. Special Options

Some options require special processing as specified in this section.

##### 4.1.3.1. Max-Age

An Inner Max-Age message field is used to indicate the maximum time a response may be cached by the client (as defined in [RFC7252]), end-to-end from the server to the client, taking into account that the option is not accessible to proxies. The Inner Max-Age SHALL be processed by OSCORE as a normal Inner option, specified in Section 4.1.1.

An Outer Max-Age message field is used to avoid unnecessary caching of error responses caused by OSCORE processing at OSCORE-unaware intermediary nodes. A server MAY set a Class U Max-Age message field with value zero to such error responses, described in Sections 7.4, 8.2, and 8.4, since these error responses are cacheable, but subsequent OSCORE requests would never create a hit in the intermediary caching it. Setting the Outer Max-Age to zero relieves the intermediary from uselessly caching responses. Successful OSCORE responses do not need to include an Outer Max-Age option since the responses appear to the OSCORE-unaware intermediary as 2.04 (Changed) responses, which are non-cacheable (see Section 4.2).

The Outer Max-Age message field is processed according to Section 4.1.2.

#### 4.1.3.2. Uri-Host and Uri-Port

When the Uri-Host and Uri-Port are set to their default values (see Section 5.10.1 [RFC7252]), they are omitted from the message (Section 5.4.4 of [RFC7252]), which is favorable both for overhead and privacy.

In order to support forward proxy operations, Proxy-Scheme, Uri-Host, and Uri-Port need to be Class U. For the use of Proxy-Uri, see Section 4.1.3.3.

Manipulation of unprotected message fields (including Uri-Host, Uri-Port, destination IP/port or request scheme) MUST NOT lead to an OSCORE message becoming verified by an unintended server. Different servers SHALL have different security contexts.

#### 4.1.3.3. Proxy-Uri

When Proxy-Uri is present, the client SHALL first decompose the Proxy-Uri value of the original CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path, and Uri-Query options according to Section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and SHALL be protected and processed as Inner options (Section 4.1.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host, and Uri-Port options as specified in Section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.1.2).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and Uri-\* options works by design for all CoAP URIs (see Section 6 of [RFC7252]). OSCORE-aware HTTP servers should not use the userinfo component of the HTTP URI (as defined in Section 3.2.1 of [RFC3986]), so that this type of replacement is possible in the presence of CoAP-to-HTTP proxies (see Section 11.2). In future specifications of cross-protocol proxying behavior using different URI structures, it is expected that the authors will create Uri-\* options that allow decomposing the Proxy-Uri, and specifying the OSCORE processing.

An example of how Proxy-Uri is processed is given here. Assume that the original CoAP message contains:

```
o Proxy-Uri = "coap://example.com/resource?q=1"
```

During OSCORE processing, Proxy-Uri is split into:

- o Proxy-Scheme = "coap"
- o Uri-Host = "example.com"
- o Uri-Port = "5683"
- o Uri-Path = "resource"
- o Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.1.1, and are thus encrypted and transported in the COSE object:

- o Uri-Path = "resource"
- o Uri-Query = "q=1"

The remaining options are composed into the Proxy-Uri included in the options part of the OSCORE message, which has value:

- o Proxy-Uri = "coap://example.com"

See Sections 6.1 and 12.6 of [RFC7252] for more details.

#### 4.1.3.4. The Block Options

Block-wise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting block-wise transfers. The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message segmentation such that each segment can be verified. The Block options, when Outer message fields, enables hop-by-hop fragmentation of the OSCORE message. Inner and Outer block processing may have different performance properties depending on the underlying transport. The end-to-end integrity of the message can be verified both in case of Inner and Outer Block-wise transfers provided all blocks are received.

##### 4.1.3.4.1. Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE. In this case the Block options SHALL be processed by OSCORE as normal Inner options (Section 4.1.1). The receiving CoAP endpoint SHALL process the OSCORE message before processing Block-wise as defined in [RFC7959].

#### 4.1.3.4.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.1.2). Note that the Outer Block options are neither encrypted nor integrity protected. As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity. Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a maximum unfragmented message size (MAX\_UNFRAGMENTED\_SIZE) considering the maximum size of message which can be handled by the endpoints. Messages exceeding this size SHOULD be fragmented by the sending endpoint using Inner Block options (Section 4.1.3.4.1).

An endpoint receiving an OSCORE message with an Outer Block option SHALL first process this option according to [RFC7959], until all blocks of the OSCORE message have been received, or the cumulated message size of the blocks exceeds MAX\_UNFRAGMENTED\_SIZE. In the former case, the processing of the OSCORE message continues as defined in this document. In the latter case the message SHALL be discarded.

Because of encryption of Uri-Path and Uri-Query, messages to the same server may, from the point of view of a proxy, look like they also target the same resource. A proxy SHOULD mitigate a potential mix-up of blocks from concurrent requests to the same server, for example using the Request-Tag processing specified in Section 3.3.2 of [I-D.ietf-core-echo-request-tag].

#### 4.1.3.5. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7641], in which case the Observe related processing can be omitted.

The support for Observe [RFC7641] with OSCORE targets the requirements on forwarding of Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs], i.e. that observations go through intermediary nodes, as illustrated in Figure 8 of [RFC7641].

Inner Observe SHALL be used to protect the value of the Observe option between the endpoints. Outer Observe SHALL be used to support forwarding by intermediary nodes.

The server SHALL include a new Partial IV (see Section 5) in responses (with or without the Observe option) to Observe

registrations, except for the first response where Partial IV MAY be omitted.

For cancellations, Section 3.6 of [RFC7641] specifies that all options MUST be identical to those in the registration request except for Observe and the set of ETag Options. For OSCORE messages, this matching is to be done to the options in the decrypted message.

[RFC7252] does not specify how the server should act upon receiving the same Token in different requests. When using OSCORE, the server SHOULD NOT remove an active observation just because it receives a request with the same Token.

Since POST with Observe is not defined, for messages with Observe, the Outer Code MUST be set to 0.05 (FETCH) for requests and to 2.05 (Content) for responses (see Section 4.2).

#### 4.1.3.5.1. Registrations and Cancellations

The Inner and Outer Observe in the request MUST contain the Observe value of the original CoAP request; 0 (registration) or 1 (cancellation).

Every time a client issues a new Observe request, a new Partial IV MUST be used (see Section 5), and so the payload and OSCORE option are changed. The server uses the Partial IV of the new request as the 'request\_piv' of all associated notifications (see Section 5.4).

Intermediaries are not assumed to have access to the OSCORE security context used by the endpoints, and thus cannot make requests or transform responses with the OSCORE option which verify at the receiving endpoint as coming from the other endpoint. This has the following consequences and limitations for Observe operations.

- o An intermediary node removing the Outer Observe 0 does not change the registration request to a request without Observe (see Section 2 of [RFC7641]). Instead other means for cancellation may be used as described in Section 3.6 of [RFC7641].
- o An intermediary node is not able to transform a normal response into an OSCORE protected Observe notification (see figure 7 of [RFC7641]) which verifies as coming from the server.
- o An intermediary node is not able to initiate an OSCORE protected Observe registration (Observe with value 0) which verifies as coming from the client. An OSCORE-aware intermediary SHALL NOT initiate registrations of observations (see Section 10). If an OSCORE-unaware proxy re-sends an old registration message from a

client this will trigger the replay protection mechanism in the server. To prevent this from resulting in the OSCORE-unaware proxy to cancel of the registration, a server MAY respond to a replayed registration request with a replay of a cached notification. Alternatively, the server MAY send a new notification.

- o An intermediary node is not able to initiate an OSCORE protected Observe cancellation (Observe with value 1) which verifies as coming from the client. An application MAY decide to allow intermediaries to cancel Observe registrations, e.g. to send Observe with value 1 (see Section 3.6 of [RFC7641]), but that can also be done with other methods, e.g. reusing the Token in a different request or sending a RST message. This is out of scope for this specification.

#### 4.1.3.5.2. Notifications

If the server accepts an Observe registration, a Partial IV MUST be included in all notifications (both successful and error), except for the first one where Partial IV MAY be omitted. To protect against replay, the client SHALL maintain a Notification Number for each Observation it registers. The Notification Number is a non-negative integer containing the largest Partial IV of the received notifications for the associated Observe registration. Further details of replay protection of notifications are specified in Section 7.4.1.

For notifications, the Inner Observe value MUST be empty (see Section 3.2 of [RFC7252]). The Outer Observe in a notification is needed for intermediary nodes to allow multiple responses to one request, and may be set to the value of Observe in the original CoAP message. The client performs ordering of notifications and replay protection by comparing their Partial IVs and SHALL ignore the outer Observe value.

If the client receives a response to an Observe request without an Inner Observe option, then it verifies the response as a non-Observe response, as specified in Section 8.4. If the client receives a response to a non-Observe request with an Inner Observe option, then it stops processing the message, as specified in Section 8.4.

A client MUST consider the notification with the highest Partial IV as the freshest, regardless of the order of arrival. In order to support existing Observe implementations the OSCORE client implementation MAY set the Observe value to the three least significant bytes of the Partial IV. Implementations need to make

sure that the notification without Partial IV is considered the oldest.

#### 4.1.3.6. No-Response

No-Response [RFC7967] is an optional feature used by the client to communicate its disinterest in certain classes of responses to a particular request. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7967].

If used, No-Response MUST be Inner. The Inner No-Response SHALL be processed by OSCORE as specified in Section 4.1.1. The Outer option SHOULD NOT be present. The server SHALL ignore the Outer No-Response option. The client MAY set the Outer No-Response value to 26 ('suppress all known codes') if the Inner value is set to 26. The client MUST be prepared to receive and discard 5.04 (Gateway Timeout) error messages from intermediaries potentially resulting from destination time out due to no response.

#### 4.1.3.7. OSCORE

The OSCORE option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed. The content in the OSCORE option is neither encrypted nor integrity protected as a whole but some part of the content of this option is protected (see Section 5.4). Nested use of OSCORE is not supported: If OSCORE processing detects an OSCORE option in the original CoAP message, then processing SHALL be stopped.

Field	E	U
Version (UDP)		x
Type (UDP)		x
Length (TCP)		x
Token Length		x
Code	x	
Message ID (UDP)		x
Token		x
Payload	x	

E = Encrypt and Integrity Protect (Inner)  
U = Unprotected (Outer)

Figure 6: Protection of CoAP Header Fields and Payload

#### 4.2. CoAP Header Fields and Payload

A summary of how the CoAP header fields and payload are protected is shown in Figure 6, including fields specific to CoAP over UDP and CoAP over TCP (marked accordingly in the table).

Most CoAP Header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response Layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP Header field Code is protected by OSCORE. Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping on or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP message into the plaintext of the COSE object (see Section 5.3). After that, the sending endpoint writes an Outer Code to the OSCORE message. With one exception (see Section 4.1.3.5) the Outer Code SHALL be set to 0.02 (POST) for requests and to 2.04 (Changed) for responses. The receiving endpoint SHALL discard the Outer Code in the OSCORE message and write the Code of the COSE object plaintext (Section 5.3) into the decrypted CoAP message.

The other currently defined CoAP Header fields are Unprotected (Class U). The sending endpoint SHALL write all other header fields of the original message into the header of the OSCORE message. The receiving endpoint SHALL write the header fields from the received OSCORE message into the header of the decrypted CoAP message.

The CoAP Payload, if present in the original CoAP message, SHALL be encrypted and integrity protected and is thus an Inner message field. The sending endpoint writes the payload of the original CoAP message into the plaintext (Section 5.3) input to the COSE object. The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the original CoAP message.

#### 4.3. Signaling Messages

Signaling messages (CoAP Code 7.00–7.31) were introduced to exchange information related to an underlying transport connection in the specific case of CoAP over reliable transports [RFC8323].



OSCORE MAY be used to protect Signaling if the endpoints for OSCORE coincide with the endpoints for the signaling message. If OSCORE is used to protect Signaling then:

- o To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.
- o Signaling messages SHALL be protected as CoAP Request messages, except in the case the Signaling message is a response to a previous Signaling message, in which case it SHALL be protected as a CoAP Response message. For example, 7.02 (Ping) is protected as a CoAP Request and 7.03 (Pong) as a CoAP response.
- o The Outer Code for Signaling messages SHALL be set to 0.02 (POST), unless it is a response to a previous Signaling message, in which case it SHALL be set to 2.04 (Changed).
- o All Signaling options, except the OSCORE option, SHALL be Inner (Class E).

NOTE: Option numbers for Signaling messages are specific to the CoAP Code (see Section 5.2 of [RFC8323]).

If OSCORE is not used to protect Signaling, Signaling messages SHALL be unaltered by OSCORE.

## 5. The COSE Object

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 Additional Data (AEAD) algorithm. The AEAD key lengths, AEAD nonce length, and maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of [RFC8152] is mandatory to implement. For AES-CCM-16-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of AEAD nonce and Common IV is 13 bytes. The maximum Sender Sequence Number is specified in Section 12.

As specified in [RFC5116], plaintext denotes the data that is to be encrypted and integrity protected, and Additional Authenticated Data (AAD) denotes the data that is to be integrity protected only.

The COSE Object SHALL be a COSE\_Encrypt0 object with fields defined as follows

- o The 'protected' field is empty.

- o The 'unprotected' field includes:
  - \* The 'Partial IV' parameter. The value is set to the Sender Sequence Number. All leading bytes of value zero SHALL be removed when encoding the Partial IV, except in the case of Partial IV of value 0 which is encoded to the byte string 0x00. This parameter SHALL be present in requests. The Partial IV SHALL be present in responses to Observe registrations (see Section 4.1.3.5.1), otherwise the Partial IV will not typically be present in responses (for one exception, see Appendix B.1.2).
  - \* The 'kid' parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and will not typically be present in responses. An example where the Sender ID is included in a response is the extension of OSCORE to group communication [I-D.ietf-core-oscore-groupcomm].
  - \* Optionally, a 'kid context' parameter (see Section 5.1). This parameter MAY be present in requests, and if so, MUST contain an ID Context (see Section 3.1). This parameter SHOULD NOT be present in responses: an example of how 'kid context' can be used in responses is given in Appendix B.2. If 'kid context' is present in the request, then the server SHALL use a security context with that ID Context when verifying the request.
- o The 'ciphertext' field is computed from the secret key (Sender Key or Recipient Key), AEAD nonce (see Section 5.2), plaintext (see Section 5.3), and the Additional Authenticated Data (AAD) (see Section 5.4) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

#### 5.1. ID Context and 'kid context'

For certain use cases, e.g. deployments where the same Sender ID is used with multiple contexts, it is possible (and sometimes necessary, see Section 3.3) for the client to use an ID Context to distinguish the security contexts (see Section 3.1). For example:

- o If the client has a unique identifier in some namespace then that identifier can be used as ID Context.
- o The ID Context may be used to add randomness into new Sender and Recipient Contexts, see Appendix B.2.

- o In case of group communication [I-D.ietf-core-oscore-groupcomm], a group identifier is used as ID Context to enable different security contexts for a server belonging to multiple groups.

The Sender ID and ID Context are used to establish the necessary input parameters and in the derivation of the security context (see Section 3.2).

Whereas the 'kid' parameter is used to transport the Sender ID, the new COSE header parameter 'kid context' is used to transport the ID Context in requests, see Figure 7.

name	label	value type	value registry	description
kid context	TBD2	bstr		Identifies the context for kid

Figure 7: Common Header Parameter 'kid context' for the COSE object

If ID Context is non-empty and the client sends a request without 'kid context' which results in an error indicating that the server could not find the security context, then the client could include the ID Context in the 'kid context' when making another request. Note that since the error is unprotected it may have been spoofed and the real response blocked by an on-path attacker.

## 5.2. AEAD Nonce

The high level design of the AEAD nonce follows Section 4.4 of [I-D.mcgregw-iv-gen], here follows the detailed construction (see Figure 8):

1. left-pad the Partial IV (PIV) with zeroes to exactly 5 bytes,
2. left-pad the Sender ID of the endpoint that generated the Partial IV (ID\_PIV) with zeroes to exactly nonce length minus 6 bytes,
3. concatenate the size of the ID\_PIV (a single byte S) with the padded ID\_PIV and the padded PIV,
4. and then XOR with the Common IV.

Note that in this specification only AEAD algorithms that use nonces equal or greater than 7 bytes are supported. The nonce construction with S, ID\_PIV, and PIV together with endpoint unique IDs and

encryption keys makes it easy to verify that the nonces used with a specific key will be unique, see Appendix D.4.

If the Partial IV is not present in a response, the nonce from the request is used. For responses that are not notifications (i.e. when there is a single response to a request), the request and the response should typically use the same nonce to reduce message overhead. Both alternatives provide all the required security properties, see Section 7.4 and Appendix D.4. The only non-Observe scenario where a Partial IV must be included in a response is when the server is unable to perform replay protection, see Appendix B.1.2. For processing instructions see Section 8.

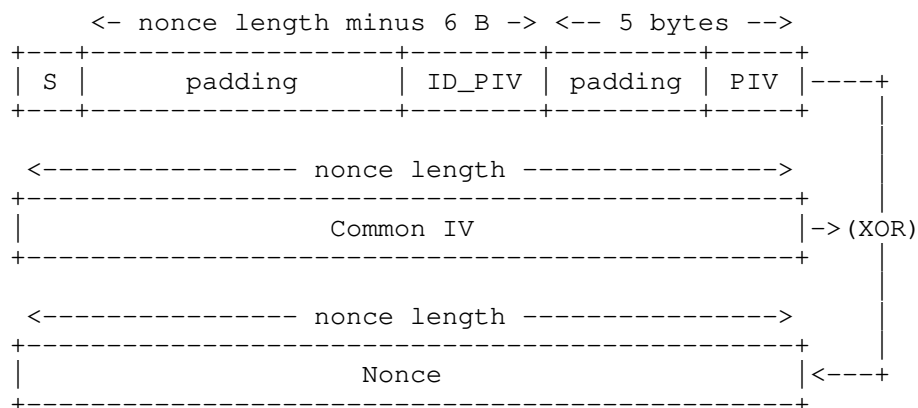


Figure 8: AEAD Nonce Formation

### 5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

- o the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and
- o all Inner option message fields (see Section 4.1.1) present in the original CoAP message (see Section 4.1). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Class E option; and
- o the Payload of original CoAP message, if present, and in that case prefixed by the one-byte Payload Marker (0xff).

NOTE: The plaintext contains all CoAP data that needs to be encrypted end-to-end between the endpoints.

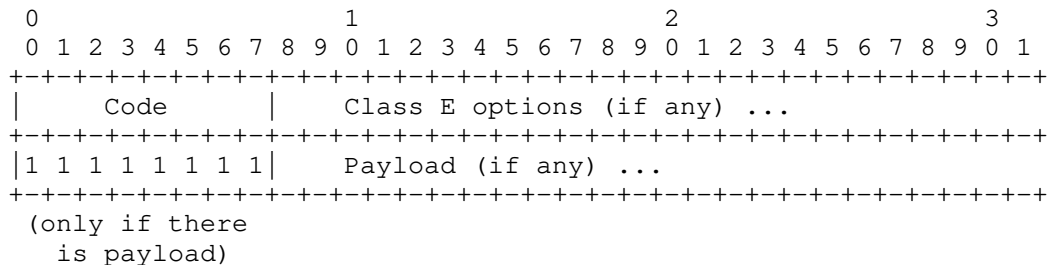


Figure 9: Plaintext

#### 5.4. Additional Authenticated Data

The `external_aad` SHALL be a CBOR array wrapped in a `bstr` object as defined below:

```

external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [ alg_aead : int / tstr ],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
]

```

where:

- o `oscore_version`: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.
- o `algorithms`: contains (for extensibility) an array of algorithms, according to this specification only containing `alg_aead`.
- o `alg_aead`: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).
- o `request_kid`: contains the value of the 'kid' in the COSE object of the request (see Section 5).
- o `request_piv`: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

- o options: contains the Class I options (see Section 4.1.2) present in the original CoAP message encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of class I option.

The `oscore_version` and `algorithms` parameters are established out-of-band and are thus never transported in OSCORE, but the `external_aad` allows to verify that they are the same in both endpoints.

NOTE: The format of the `external_aad` is for simplicity the same for requests and responses, although some parameters, e.g. `request_kid`, need not be integrity protected in all requests.

The Additional Authenticated Data (AAD) is composed from the `external_aad` as described in Section 5.3 of [RFC8152]:

```
AAD = Enc_structure = [ "Encrypt0", h'', external_aad ]
```

The following is an example of AAD constructed using AEAD Algorithm = AES-CCM-16-64-128 (10), `request_kid` = 0x00, `request_piv` = 0x25 and no Class I options:

- o `oscore_version`: 0x01 (1 byte)
- o `algorithms`: 0x810a (2 bytes)
- o `request_kid`: 0x00 (1 byte)
- o `request_piv`: 0x25 (1 byte)
- o `options`: 0x (0 bytes)
- o `aad_array`: 0x8501810a4100412540 (9 bytes)
- o `external_aad`: 0x498501810a4100412540 (10 bytes)
- o `AAD`: 0x8368456e63727970743040498501810a4100412540 (21 bytes)

Note that the AAD consists of a fixed string of 11 bytes concatenated with the `external_aad`.

## 6. OSCORE Header Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [RFC8152] uses CBOR to create compact encoding of signed and encrypted data. COSE is however constructed to support a large number of different stateless use cases, and is not fully

optimized for use as a stateful security protocol, leading to a larger than necessary message expansion. In this section, we define a stateless header compression mechanism, simply removing redundant information from the COSE objects, which significantly reduces the per-packet overhead. The result of applying this mechanism to a COSE object is called the "compressed COSE object".

The COSE\_Encrypt0 object used in OSCORE is transported in the OSCORE option and in the Payload. The Payload contains the Ciphertext of the COSE object. The headers of the COSE object are compactly encoded as described in the next section.

### 6.1. Encoding of the OSCORE Option Value

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 as follows:

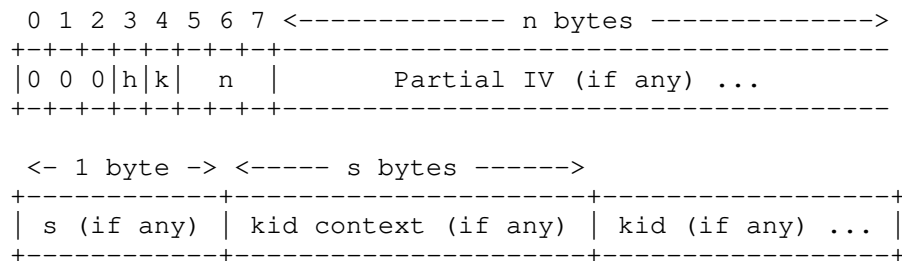


Figure 10: The OSCORE Option Value

- o The first byte, containing the OSCORE flag bits, encodes the following set of bits and the length of the Partial IV parameter:
  - \* The three least significant bits encode the Partial IV length n. If n = 0 then the Partial IV is not present in the compressed COSE object. The values n = 6 and n = 7 are reserved.
  - \* The fourth least significant bit is the 'kid' flag, k: it is set to 1 if the kid is present in the compressed COSE object.
  - \* The fifth least significant bit is the 'kid context' flag, h: it is set to 1 if the compressed COSE object contains a 'kid context' (see Section 5.1).
  - \* The sixth to eighth least significant bits are reserved for future use. These bits SHALL be set to zero when not in use. According to this specification, if any of these bits are set

to 1 the message is considered to be malformed and decompression fails as specified in item 2 of Section 8.2.

The flag bits are registered in the OSCORE Flag Bits registry specified in Section 13.7.

- o The following  $n$  bytes encode the value of the Partial IV, if the Partial IV is present ( $n > 0$ ).
- o The following 1 byte encode the length of the 'kid context' (Section 5.1)  $s$ , if the 'kid context' flag is set ( $h = 1$ ).
- o The following  $s$  bytes encode the 'kid context', if the 'kid context' flag is set ( $h = 1$ ).
- o The remaining bytes encode the value of the 'kid', if the 'kid' is present ( $k = 1$ ).

Note that the 'kid' MUST be the last field of the OSCORE option value, even in case reserved bits are used and additional fields are added to it.

The length of the OSCORE option thus depends on the presence and length of Partial IV, 'kid context', 'kid', as specified in this section, and on the presence and length of the other parameters, as defined in the separate documents.

## 6.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object.

## 6.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for requests and responses. The examples assume the COSE\_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the full CoAP unprotected message, as well as the full security context, is not reported in the examples, 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 6, divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, and Partial IV = 0x05



Before compression (24 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (17 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x090525 (3 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

2. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, and Partial IV = 0x00

Before compression (23 bytes):

```
[
  h'',
  { 4:h'', 6:h'00' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x0900 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

3. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, Partial IV = 0x05, and kid context = 0x44616c656b

Before compression (30 bytes):

```
[
  h'',
  { 4:h'', 6:h'05', 8:h'44616c656b' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (22 bytes):

Flag byte: 0b00011001 = 0x19 (1 byte)

Option Value: 0x19050544616c656b (8 bytes)

Payload: 0xae a0155667924dff8a24e4cb35b9 (14 bytes)

4. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and no Partial IV

Before compression (18 bytes):

```
[
  h'',
  {},
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00 (1 byte)

Option Value: 0x (0 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

5. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and Partial IV = 0x07

Before compression (21 bytes):

```
[
  h'',
  { 6:h'07' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01 (1 byte)

Option Value: 0x0107 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

## 7. Message Binding, Sequence Numbers, Freshness, and Replay Protection

### 7.1. Message Binding

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised intermediaries, OSCORE binds responses to the requests by including the 'kid' and Partial IV of the request in the AAD of the response. The server therefore needs to store the 'kid' and Partial IV of the request until all responses have been sent.

### 7.2. Sequence Numbers

An AEAD nonce MUST NOT be used more than once per AEAD key. The uniqueness of (key, nonce) pairs is shown in Appendix D.4, and in particular depends on a correct usage of Partial IVs (which encode the Sender Sequence Numbers, see Section 5). If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

#### 7.2.1. Maximum Sequence Number

The maximum Sender Sequence Number is algorithm dependent (see Section 12), and SHALL be less than  $2^{40}$ . If the Sender Sequence Number exceeds the maximum, the endpoint MUST NOT process any more messages with the given Sender Context. If necessary, the endpoint SHOULD acquire a new security context before this happens. The latter is out of scope of this document.

### 7.3. Freshness

For requests, OSCORE provides only the guarantee that the request is not older than the security context. For applications having stronger demands on request freshness (e.g., control of actuators), OSCORE needs to be augmented with mechanisms providing freshness, for example as specified in [I-D.ietf-core-echo-request-tag].

Assuming an honest server (see Appendix D), the message binding guarantees that a response is not older than its request. For responses that are not notifications (i.e. when there is a single response to a request), this gives absolute freshness. For notifications, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly re-register the observation. Note that the message binding does not guarantee that misbehaving server created the response before receiving the request, i.e. it does not verify server aliveness.

For requests and notifications, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of requests or responses.

#### 7.4. Replay Protection

In order to protect from replay of requests, the server's Recipient Context includes a Replay Window. A server SHALL verify that a Partial IV = Sender Sequence Number received in the COSE object has not been received before. If this verification fails, the server SHALL stop processing the message, and MAY optionally respond with a 4.01 (Unauthorized) error message. Also, the server MAY set an Outer Max-Age option with value zero, to inform any intermediary that the response is not to be cached. The diagnostic payload MAY contain the "Replay detected" string. The size and type of the Replay Window depends on the use case and the protocol with which the OSCORE message is transported. In case of reliable and ordered transport from endpoint to endpoint, e.g. TCP, the server MAY just store the last received Partial IV and require that newly received Partial IVs equals the last received Partial IV + 1. However, in case of mixed reliable and unreliable transports and where messages may be lost, such a replay mechanism may be too restrictive and the default replay window be more suitable (see Section 3.2.2).

Responses (with or without Partial IV) are protected against replay as they are bound to the request and the fact that only a single response is accepted. Note that the Partial IV is not used for replay protection in this case.

The operation of validating the Partial IV and updating the replay protection MUST be atomic.

##### 7.4.1. Replay Protection of Notifications

The following applies additionally when Observe is supported.

The Notification Number is initialized to the Partial IV of the first successfully verified notification in response to the registration request. A client MUST only accept at most one Observe notifications without Partial IV, and treat it as the oldest notification received. A client receiving a notification containing a Partial IV SHALL compare the Partial IV with the Notification Number associated to that Observe registration. The client MUST stop processing notifications with a Partial IV which has been previously received. Applications MAY decide that a client only processes notifications which have greater Partial IV than the Notification Number.

If the verification of the response succeeds, and the received Partial IV was greater than the Notification Number then the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

#### 7.5. Losing Part of the Context State

To prevent reuse of an AEAD nonce with the same AEAD key, or from accepting replayed messages, an endpoint needs to handle the situation of losing rapidly changing parts of the context, such as the Sender Sequence Number, and Replay Window. These are typically stored in RAM and therefore lost in the case of e.g. an unplanned reboot. There are different alternatives to recover, for example:

1. The endpoints can reuse an existing Security Context after updating the mutable parts of the security context (Sender Sequence Number, and Replay Window). This requires that the mutable parts of the security context are available throughout the lifetime of the device, or that the device can establish safe security context after loss of mutable security context data. Examples is given based on careful use of non-volatile memory, see Appendix B.1.1, and additionally the use of the Echo option, see Appendix B.1.2. If an endpoint makes use of a partial security context stored in non-volatile memory, it MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously received messages.
2. The endpoints can reuse an existing shared Master Secret and derive new Sender and Recipient Contexts, see Appendix B.2 for an example. This typically requires a good source of randomness.
3. The endpoints can use a trusted-third party assisted key establishment protocol such as [I-D.ietf-ace-oscore-profile]. This requires the execution of three-party protocol and may require a good source of randomness.
4. The endpoints can run a key exchange protocol providing forward secrecy resulting in a fresh Master Secret, from which an entirely new Security Context is derived. This requires a good source of randomness, and additionally, the transmission and processing of the protocol may have a non-negligible cost, e.g. in terms of power consumption.

The endpoints need to be configured with information about which method is used. The choice of method may depend on capabilities of the devices deployed and the solution architecture. Using a key exchange protocol is necessary for deployments that require forward secrecy.

## 8. Processing

This section describes the OSCORE message processing. Additional processing for Observe or Block-wise are described in subsections.

Note that, analogously to [RFC7252] where the Token and source/destination pair are used to match a response with a request, both endpoints MUST keep the association (Token, {Security Context, Partial IV of the request}), in order to be able to find the Security Context and compute the AAD to protect or verify the response. The association MAY be forgotten after it has been used to successfully protect or verify the response, with the exception of Observe processing, where the association MUST be kept as long as the Observation is active.

The processing of the Sender Sequence Number follows the procedure described in Section 3 of [I-D.mcgregor-iv-gen].

### 8.1. Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to create an OSCORE request:

1. Retrieve the Sender Context associated with the target resource.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV as described in Section 5.2.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

### 8.2. Verifying the Request

A server receiving a request containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

2. Decompress the COSE Object (Section 6) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter, additionally using the 'kid context', if present. If either the decompression or the COSE message fails to decode, or the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, then the server SHALL stop processing the request.
  - \* If either the decompression or the COSE message fails to decode, the server MAY respond with a 4.02 (Bad Option) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Failed to decode COSE".
  - \* If the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, the server MAY respond with a 4.01 (Unauthorized) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Security context not found".
3. Verify that the 'Partial IV' has not been received before using the Replay Window, as described in Section 7.4.
4. Compose the Additional Authenticated Data, as described in Section 5.4.
5. Compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.
6. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)
  - \* If decryption fails, the server MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the "Decryption failed" string.
  - \* If decryption succeeds, update the Replay Window, as described in Section 7.
7. Add decrypted Code, options, and payload to the decrypted request. The OSCORE option is removed.
8. The decrypted CoAP request is processed according to [RFC7252].

#### 8.2.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

#### 8.3. Protecting the Response

If a CoAP response is generated in response to an OSCORE request, the server SHALL perform the following steps to create an OSCORE response. Note that CoAP error responses derived from CoAP processing (step 8 in Section 8.2) are protected, as well as successful CoAP responses, while the OSCORE errors (steps 2, 3, and 6 in Section 8.2) do not follow the processing below, but are sent as simple CoAP responses, without OSCORE processing.

1. Retrieve the Sender Context in the Security Context associated with the Token.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Compute the AEAD nonce as described in Section 5.2:
  - \* Either use the AEAD nonce from the request, or
  - \* Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6. If the AEAD nonce was constructed from a new Partial IV, this Partial IV MUST be included in the message. If the AEAD nonce from the request was used, the Partial IV MUST NOT be included in the message.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

##### 8.3.1. Supporting Observe

If Observe is supported, insert the following step between step 2 and 3 of Section 8.3:



- A. If the response is an observe notification:
  - o If the response is the first notification:
    - \* compute the AEAD nonce as described in Section 5.2:
      - + Either use the AEAD nonce from the request, or
      - + Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
  - Then go to 4.
- o If the response is not the first notification:
  - \* encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV, then go to 4.

#### 8.4. Verifying the Response

A client receiving a response containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, ETag Outer option is discarded, as well as Max-Age Outer option.
2. Retrieve the Recipient Context in the Security Context associated with the Token. Decompress the COSE Object (Section 6). If either the decompression or the COSE message fails to decode, then go to 8.
3. Compose the Additional Authenticated Data, as described in Section 5.4.
4. Compute the AEAD nonce
  - \* If the Partial IV is not present in the response, the AEAD nonce from the request is used.
  - \* If the Partial IV is present in the response, compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

5. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.) If decryption fails, then go to 8.
6. Add decrypted Code, options and payload to the decrypted request. The OSCORE option is removed.
7. The decrypted CoAP response is processed according to [RFC7252].
8. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response.

#### 8.4.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

- A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

#### 8.4.2. Supporting Observe

If Observe is supported:

Insert the following step between step 5 and step 6:

- A. If the request was an Observe registration, then:
  - o If the Partial IV is not present in the response, and Inner Observe is present, and the AEAD nonce from the request was already used once, then go to 8.
  - o If the Partial IV is present in the response and Inner Observe is present, then follow the processing described in Section 4.1.3.5.2 and Section 7.4.1, then:
    - \* initialize the Notification Number (if first successfully verified notification), or
    - \* overwrite the Notification Number (if the received Partial IV was greater than the Notification Number).

Replace step 8 of Section 8.4 with:

- B. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response. An error condition occurring while processing a response to an observation request does

not cancel the observation. A client MUST NOT react to failure by re-registering the observation immediately.

## 9. Web Linking

The use of OSCORE MAY be indicated by a target attribute "osc" in a web link [RFC8288] to a resource, e.g. using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "osc" attribute is a hint indicating that the destination of that link is only accessible using OSCORE, and unprotected access to it is not supported. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCORE.

A value MUST NOT be given for the "osc" attribute; any present value MUST be ignored by parsers. The "osc" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

The example in Figure 11 shows a use of the "osc" attribute: the client does resource discovery on a server, and gets back a list of resources, one of which includes the "osc" attribute indicating that the resource is protected with OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core
RES: 2.05 Content
    </sensors/temp>;osc,
    </sensors/light>;if="sensor"
```

Figure 11: The web link

## 10. CoAP-to-CoAP Forwarding Proxy

CoAP is designed for proxy operations (see Section 5.7 of [RFC7252]).

OSCORE is designed to work with OSCORE-unaware CoAP proxies. Security requirements for forwarding are listed in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. Proxy processing of the (Outer) Proxy-Uri option works as defined in [RFC7252]. Proxy processing of the (Outer) Block options works as defined in [RFC7959].

However, not all CoAP proxy operations are useful:

- o Since a CoAP response is only applicable to the original CoAP request, caching is in general not useful. In support of existing

proxies, OSCORE uses the outer Max-Age option, see Section 4.1.3.1.

- o Proxy processing of the (Outer) Observe option as defined in [RFC7641] is specified in Section 4.1.3.5.

Optionally, a CoAP proxy MAY detect OSCORE and act accordingly. An OSCORE-aware CoAP proxy:

- o SHALL bypass caching for the request if the OSCORE option is present
- o SHOULD avoid caching responses to requests with an OSCORE option

In the case of Observe (see Section 4.1.3.5) the OSCORE-aware CoAP proxy:

- o SHALL NOT initiate an Observe registration
- o MAY verify the order of notifications using Partial IV rather than the Observe option

## 11. HTTP Operations

The CoAP request/response model may be mapped to HTTP and vice versa as described in Section 10 of [RFC7252]. The HTTP-CoAP mapping is further detailed in [RFC8075]. This section defines the components needed to map and transport OSCORE messages over HTTP hops. By mapping between HTTP and CoAP and by using cross-protocol proxies OSCORE may be used end-to-end between e.g. an HTTP client and a CoAP server. Examples are provided at the end of the section.

### 11.1. The HTTP OSCORE Header Field

The HTTP OSCORE Header Field (see Section 13.4) is used for carrying the content of the CoAP OSCORE option when transporting OSCORE messages over HTTP hops.

The HTTP OSCORE header field is only used in POST requests and 200 (OK) responses. When used, the HTTP header field Content-Type is set to 'application/oscore' (see Section 13.5) indicating that the HTTP body of this message contains the OSCORE payload (see Section 6.2). No additional semantics is provided by other message fields.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP OSCORE header field value is as follows.

base64url-char = ALPHA / DIGIT / "-" / "\_"

OSCORE = 2\*base64url-char

The HTTP OSCORE header field is not appropriate to list in the Connection header field (see Section 6.1 of [RFC7230]) since it is not hop-by-hop. OSCORE messages are generally not useful when served from cache (i.e., they will generally be marked Cache-Control: no-cache) and so interaction with Vary is not relevant (Section 7.1.4 of [RFC7231]). Since the HTTP OSCORE header field is critical for message processing, moving it from headers to trailers renders the message unusable in case trailers are ignored (see Section 4.1 of [RFC7230]).

Intermediaries are in general not allowed to insert, delete, or modify the OSCORE header. Changes to the HTTP OSCORE header field will in general violate the integrity of the OSCORE message resulting in an error. For the same reason the HTTP OSCORE header field is in general not preserved across redirects.

Since redirects are not defined in the mappings between HTTP and CoAP [RFC8075][RFC7252], a number of conditions need to be fulfilled for redirects to work. For CoAP client to HTTP server, such conditions include:

- o the CoAP-to-HTTP proxy follows the redirect, instead of the CoAP client as in the HTTP case
- o the CoAP-to-HTTP proxy copies the HTTP OSCORE header field and body to the new request
- o the target of the redirect has the necessary OSCORE security context required to decrypt and verify the message

Since OSCORE requires HTTP body to be preserved across redirects, the HTTP server is RECOMMENDED to reply with 307 or 308 instead of 301 or 302.

For the case of HTTP client to CoAP server, although redirect is not defined for CoAP servers [RFC7252], an HTTP client receiving a redirect should generate a new OSCORE request for the server it was redirected to.

## 11.2. CoAP-to-HTTP Mapping

Section 10.1 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP cross-protocol mapping process. The additional rules for OSCORE messages are:

- o The HTTP OSCORE header field value is set to
  - \* AA if the CoAP OSCORE option is empty, otherwise
  - \* the value of the CoAP OSCORE option (Section 6.1) in base64url (Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The HTTP Content-Type is set to 'application/oscore' (see Section 13.5), independent of CoAP Content-Format.

### 11.3. HTTP-to-CoAP Mapping

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. The additional rules for HTTP messages with the OSCORE header field are:

- o The CoAP OSCORE option is set as follows:
  - \* empty if the value of the HTTP OSCORE header field is a single zero byte (0x00) represented by AA, otherwise
  - \* the value of the HTTP OSCORE header field decoded from base64url (Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The CoAP Content-Format option is omitted, the content format for OSCORE (Section 13.6) MUST NOT be used.

### 11.4. HTTP Endpoints

Restricted to subsets of HTTP and CoAP supporting a bijective mapping, OSCORE can be originated or terminated in HTTP endpoints.

The sending HTTP endpoint uses [RFC8075] to translate the HTTP message into a CoAP message. The CoAP message is then processed with OSCORE as defined in this document. The OSCORE message is then mapped to HTTP as described in Section 11.2 and sent in compliance with the rules in Section 11.1.

The receiving HTTP endpoint maps the HTTP message to a CoAP message using [RFC8075] and Section 11.3. The resulting OSCORE message is processed as defined in this document. If successful, the plaintext CoAP message is translated to HTTP for normal processing in the endpoint.

### 11.5. Example: HTTP Client and CoAP Server

This section is giving an example of how a request and a response between an HTTP client and a CoAP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps.

Mapping and notation here is based on "Simple Form" (Section 5.4.1 of [RFC8075]).

[HTTP request -- Before client object security processing]

```
GET http://proxy.url/hc/?target_uri=coap://server.url/orders
HTTP/1.1
```

[HTTP request -- HTTP Client to Proxy]

```
POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- Proxy to CoAP Server]

```
POST coap://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- After server object security processing]

```
GET coap://server.url/orders
```

[CoAP response -- Before server object security processing]

```
2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!
```

[CoAP response -- CoAP Server to Proxy]

```
2.04 Changed
OSCORE: [empty]
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- Proxy to HTTP Client]

```
HTTP/1.1 200 OK
Content-Type: application/oscore
OSCORE: AA
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- After client object security processing]

```
HTTP/1.1 200 OK
Content-Type: text/plain
Body: Exterminate! Exterminate!
```

Note that the HTTP Status Code 200 in the next-to-last message is the mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200 in the last message is the mapping of the CoAP Code 2.05 (Content), which was encrypted within the compressed COSE object carried in the Body of the HTTP response.

#### 11.6. Example: CoAP Client and HTTP Server

This section is giving an example of how a request and a response between a CoAP client and an HTTP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps

[CoAP request -- Before client object security processing]

```
GET coap://proxy.url/
Proxy-Uri=http://server.url/orders
```

[CoAP request -- CoAP Client to Proxy]

```
POST coap://proxy.url/
Proxy-Uri=http://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- Proxy to HTTP Server]

```
POST http://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- After server object security processing]

```
GET http://server.url/orders HTTP/1.1
```



[HTTP response -- Before server object security processing]

HTTP/1.1 200 OK  
Content-Type: text/plain  
Body: Exterminate! Exterminate!

[HTTP response -- HTTP Server to Proxy]

HTTP/1.1 200 OK  
Content-Type: application/oscore  
OSCORE: AA  
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- Proxy to CoAP Client]

2.04 Changed  
OSCORE: [empty]  
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- After client object security processing]

2.05 Content  
Content-Format: 0  
Payload: Exterminate! Exterminate!

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05 (Content) in the last message is the value that was encrypted within the compressed COSE object carried in the Body of the HTTP response.

## 12. Security Considerations

An overview of the security properties is given in Appendix D.

### 12.1. End-to-end Protection

In scenarios with intermediary nodes such as proxies or gateways, transport layer security such as (D)TLS only protects data hop-by-hop. As a consequence, the intermediary nodes can read and modify any information. The trust model where all intermediary nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

(D)TLS protects hop-by-hop the entire message. OSCORE protects end-to-end all information that is not required for proxy operations (see Section 4). (D)TLS and OSCORE can be combined, thereby enabling end-to-end security of the message payload, in combination with hop-by-hop protection of the entire message, during transport between endpoint and intermediary node. In particular when OSCORE is used with HTTP, the additional TLS protection of HTTP hops is RECOMMENDED, e.g. between an HTTP endpoint and a proxy translating between HTTP and CoAP.

Applications need to consider that certain message fields and messages types are not protected end-to-end and may be spoofed or manipulated. The consequences of unprotected message fields are analyzed in Appendix D.5.

## 12.2. Security Context Establishment

The use of COSE\_Encrypt0 and AEAD to protect messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common Master Secret and unique Sender IDs. The necessary input parameters may be pre-established or obtained using a key establishment protocol augmented with establishment of Sender/Recipient ID, such as a key exchange protocol or the OSCORE profile of the ACE framework [I-D.ietf-ace-oscore-profile]. Such a procedure must ensure that the requirements of the security context parameters for the intended use are complied with (see Section 3.3) and also in error situations. While recipient IDs are allowed to coincide between different security contexts (see Section 3.3), this may cause a server to process multiple verifications before finding the right security context or rejecting a message. Considerations for deploying OSCORE with a fixed Master Secret are given in Appendix B.

## 12.3. Master Secret

OSCORE uses HKDF [RFC5869] and the established input parameters to derive the security context. The required properties of the security context parameters are discussed in Section 3.3, in this section we focus on the Master Secret. HKDF denotes in this specification the composition of the expand and extract functions as defined in [RFC5869] and the Master Secret is used as Input Key Material (IKM).

Informally, HKDF takes as source an IKM containing some good amount of randomness but not necessarily distributed uniformly (or for which an attacker has some partial knowledge) and derive from it one or more cryptographically strong secret keys [RFC5869].

Therefore, the main requirement for the OSCORE Master Secret, in addition to being secret, is that it has a good amount of randomness. The selected key establishment schemes must ensure that the necessary properties for the Master Secret are fulfilled. For pre-shared key deployments and key transport solutions such as [I-D.ietf-ace-oscore-profile], the Master Secret can be generated offline using a good random number generator. Randomness requirements for security are described in [RFC4086].

#### 12.4. Replay Protection

Replay attacks need to be considered in different parts of the implementation. Most AEAD algorithms require a unique nonce for each message, for which the sender sequence numbers in the COSE message field 'Partial IV' is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport, it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. An adversary may try to induce a device reboot for the purpose of replaying a message (see Section 7.5).

Note that sharing a security context between servers may open up for replay attacks, for example if the replay windows are not synchronized.

#### 12.5. Client Aliveness

A verified OSCORE request enables the server to verify the identity of the entity who generated the message. However, it does not verify that the client is currently involved in the communication, since the message may be a delayed delivery of a previously generated request which now reaches the server. To verify the aliveness of the client the server may use the Echo option in the response to a request from the client (see [I-D.ietf-core-echo-request-tag]).

#### 12.6. Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number is  $2^{40} - 1$ , or any algorithm specific lower limit, after which a new security context must be generated. The mechanism to build the AEAD nonce (Section 5.2) assumes that the nonce is at least 56 bits, and the Partial IV is at most 40 bits. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM\*. AEAD algorithms that require unpredictable nonces are not supported.

In order to prevent cryptanalysis when the same plaintext is repeatedly encrypted by many different users with distinct AEAD keys, the AEAD nonce is formed by mixing the sequence number with a secret per-context initialization vector (Common IV) derived along with the keys (see Section 3.1 of [RFC8152]), and by using a Master Salt in the key derivation (see [MF00] for an overview). The Master Secret, Sender Key, Recipient Key, and Common IV must be secret, the rest of the parameters may be public. The Master Secret must have a good amount of randomness (see Section 12.3).

The ID Context, Sender ID, and Partial IV are always at least implicitly integrity protected, as manipulation leads to the wrong nonce or key being used and therefore results in decryption failure.

#### 12.7. Message Segmentation

The Inner Block options enable the sender to split large messages into OSCORE-protected blocks such that the receiving endpoint can verify blocks before having received the complete message. The Outer Block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the Inner Block options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

#### 12.8. Privacy Considerations

Privacy threats executed through intermediary nodes are considerably reduced by means of OSCORE. End-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

The unprotected options (Figure 5) may reveal privacy sensitive information, see Appendix D.5. CoAP headers sent in plaintext allow, for example, matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis. OSCORE does not provide protection for HTTP header fields which are not both CoAP-mappable and class E. The HTTP message fields which are visible to on-path entity are only used for the purpose of transporting the OSCORE message, whereas the application layer message is encoded in CoAP and encrypted.

COSE message fields, i.e. the OSCORE option, may reveal information about the communicating endpoints. E.g. 'kid' and 'kid context',

which are intended to help the server find the right context, may reveal information about the client. Tracking 'kid' and 'kid context' to one server may be used for correlating requests from one client.

Unprotected error messages reveal information about the security state in the communication between the endpoints. Unprotected signaling messages reveal information about the reliable transport used on a leg of the path. Using the mechanisms described in Section 7.5 may reveal when a device goes through a reboot. This can be mitigated by the device storing the precise state of sender sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the message. Applications may use a padding scheme to protect against traffic analysis.

### 13. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[this document]]" with the RFC number of this specification.

Note to IANA: Please note all occurrences of "TBD1" in this specification should be assigned the same number.

#### 13.1. COSE Header Parameters Registry

The 'kid context' parameter is added to the "COSE Header Parameters Registry":

- o Name: kid context
- o Label: TBD2
- o Value Type: bstr
- o Value Registry:
- o Description: Identifies the context for 'kid'
- o Reference: Section 5.1 of this document

Note to IANA: Label assignment in (Integer value between 1 and 255) is requested. (RFC Editor: Delete this note after IANA assignment)

## 13.2. CoAP Option Numbers Registry

The OSCORE option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD1	OSCORE	[[this document]]

Note to IANA: Label assignment in (Integer value between 0 and 12) is requested. We also request Expert review if possible, to make sure a correct number for the option is selected (RFC Editor: Delete this note after IANA assignment)

Furthermore, the following existing entries in the CoAP Option Numbers registry are updated with a reference to the document specifying OSCORE processing of that option:

Number	Name	Reference
1	If-Match	[RFC7252] [[this document]]
3	Uri-Host	[RFC7252] [[this document]]
4	ETag	[RFC7252] [[this document]]
5	If-None-Match	[RFC7252] [[this document]]
6	Observe	[RFC7641] [[this document]]
7	Uri-Port	[RFC7252] [[this document]]
8	Location-Path	[RFC7252] [[this document]]
11	Uri-Path	[RFC7252] [[this document]]
12	Content-Format	[RFC7252] [[this document]]
14	Max-Age	[RFC7252] [[this document]]
15	Uri-Query	[RFC7252] [[this document]]
17	Accept	[RFC7252] [[this document]]
20	Location-Query	[RFC7252] [[this document]]
23	Block2	[RFC7959] [RFC8323] [[this document]]
27	Block1	[RFC7959] [RFC8323] [[this document]]
28	Size2	[RFC7959] [[this document]]
35	Proxy-Uri	[RFC7252] [[this document]]
39	Proxy-Scheme	[RFC7252] [[this document]]
60	Size1	[RFC7252] [[this document]]
258	No-Response	[RFC7967] [[this document]]

Future additions to the CoAP Option Numbers registry need to provide a reference to the document where the OSCORE processing of that CoAP Option is defined.

### 13.3. CoAP Signaling Option Numbers Registry

The OSCORE option is added to the CoAP Signaling Option Numbers registry:

Applies to	Number	Name	Reference
7.xx (all)	TBD1	OSCORE	[[this document]]

Note to IANA: The value in the "Number" field is the same value that's being assigned to the new Option Number. Please make sure TBD1 is not the same as any value in Numbers for any existing entry in the CoAP Signaling Option Numbers registry (at the time of writing this, that means make sure TBD1 is not 2 or 4) (RFC Editor: Delete this note after IANA assignment)

### 13.4. Header Field Registrations

The HTTP OSCORE header field is added to the Message Headers registry:

Header Field Name	Protocol	Status	Reference
OSCORE	http	standard	[[this document]], Section 11.1

### 13.5. Media Type Registrations

This section registers the 'application/oscore' media type in the "Media Types" registry. These media types are used to indicate that the content is an OSCORE message. The OSCORE body cannot be understood without the OSCORE header field value and the security context.

Type name: application

Subtype name: oscore

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[This document]].

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: IoT applications sending security content over HTTP(S) transports.

Fragment identifier considerations: N/A

Additional information:

- \* Deprecated alias names for this type: N/A

- \* Magic number(s): N/A

- \* File extension(s): N/A

- \* Macintosh file type code(s): N/A

Person & email address to contact for further information:  
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

Change Controller: IESG

Provisional registration? No



### 13.6. CoAP Content-Formats Registry

Note to IANA: ID assignment in the 10000-64999 range is requested.  
(RFC Editor: Delete this note after IANA assignment)

This section registers the media type 'application/oscore' media type in the "CoAP Content-Formats" registry. This Content-Format for the OSCORE payload is defined for potential future use cases and SHALL NOT be used in the OSCORE message. The OSCORE payload cannot be understood without the OSCORE option value and the security context.

Media Type	Encoding	ID	Reference
application/oscore		TBD3	[[this document]]

### 13.7. OSCORE Flag Bits Registry

This document defines a sub-registry for the OSCORE flag bits within the "CoRE Parameters" registry. The name of the sub-registry is "OSCORE Flag Bits". The registry should be created with the Expert Review policy. Guidelines for the experts are provided in Section 13.8.

The columns of the registry are:

- o bit position: This indicates the position of the bit in the set of OSCORE flag bits, starting at 0 for the most significant bit. The bit position must be an integer or a range of integers, in the range 0 to 63.
- o name: The name is present to make it easier to refer to and discuss the registration entry. The value is not used in the protocol. Names are to be unique in the table.
- o description: This contains a brief description of the use of the bit.
- o specification: This contains a pointer to the specification defining the entry.

The initial contents of the registry can be found in the table below. The specification column for all rows in that table should be this document. The entries with Bit Position of 0 and 1 are to be marked as 'Reserved'. The entry with Bit Position of 1 is going to be specified in a future document, and will be used to expand the space

for the OSCORE flag bits in Section 6.1, so that entries 8-63 of the registry are defined.

Bit Position	Name	Description	Specification
0	Reserved		
1	Reserved		
2	Unassigned		
3	Kid Context Flag	Set to 1 if 'kid context' is present in the compressed COSE object	[[this document]]
4	Kid Flag	Set to 1 if kid is present in the compressed COSE object	[[this document]]
5-7	Partial IV Length	Encodes the Partial IV length; can have value 0 to 5	[[this document]]
8-63	Unassigned		

### 13.8. Expert Review Instructions

The expert reviewers for the registry defined in this document are expected to ensure that the usage solves a valid use case that could not be solved better in a different way, that it is not going to duplicate one that is already registered, and that the registered point is likely to be used in deployments. They are furthermore expected to check the clarity of purpose and use of the requested code points. Experts should take into account the expected usage of entries when approving point assignment, and the length of the encoded value should be weighed against the number of code points left that encode to that size and the size of device it will be used on. Experts should block registration for entries 8-63 until these points are defined (i.e. until the mechanism for the OSCORE flag bits expansion via bit 1 is specified).

## 14. References

### 14.1. 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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

- [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>>.
- [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>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8323] Bormann, C., Lemay, S., Tschafenig, 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>>.

## 14.2. Informative References

- [I-D.bormann-6lo-coap-802-15-ie]  
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.
- [I-D.hartke-core-e2e-security-reqs]  
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.
- [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-22 (work in progress), March 2019.
- [I-D.ietf-ace-oscore-profile]  
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-07 (work in progress), February 2019.
- [I-D.ietf-cbor-cddl]  
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-07 (work in progress), February 2019.
- [I-D.ietf-core-echo-request-tag]  
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [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-03 (work in progress), October 2018.
- [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.

- [I-D.mcgrew-iv-gen] McGrew, D., "Generation of Deterministic Initialization Vectors (IVs) and Nonces", draft-mcgrew-iv-gen-03 (work in progress), October 2013.
- [MF00] McGrew, D. and S. Fluhrer, "Attacks on Encryption of Redundant Plaintext and Implications on Internet Security", the Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag. , 2000.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [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>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/info/rfc5116>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [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>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [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>>.

## Appendix A. Scenario Examples

This section gives examples of OSCORE, targeting scenarios in Section 2.2.1.1 of [I-D.hartke-core-e2e-security-reqs]. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the (compressed) COSE message format.

### A.1. Secure Access to Sensor

This example illustrates a client requesting the alarm status from a server.

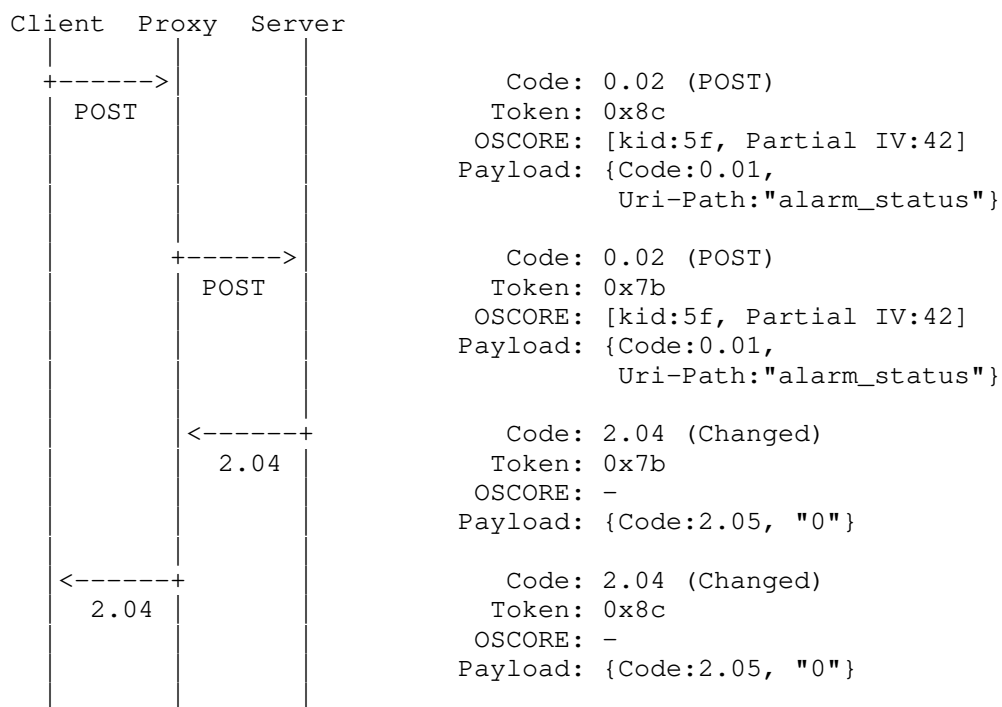


Figure 12: Secure Access to Sensor. Square brackets [ ... ] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

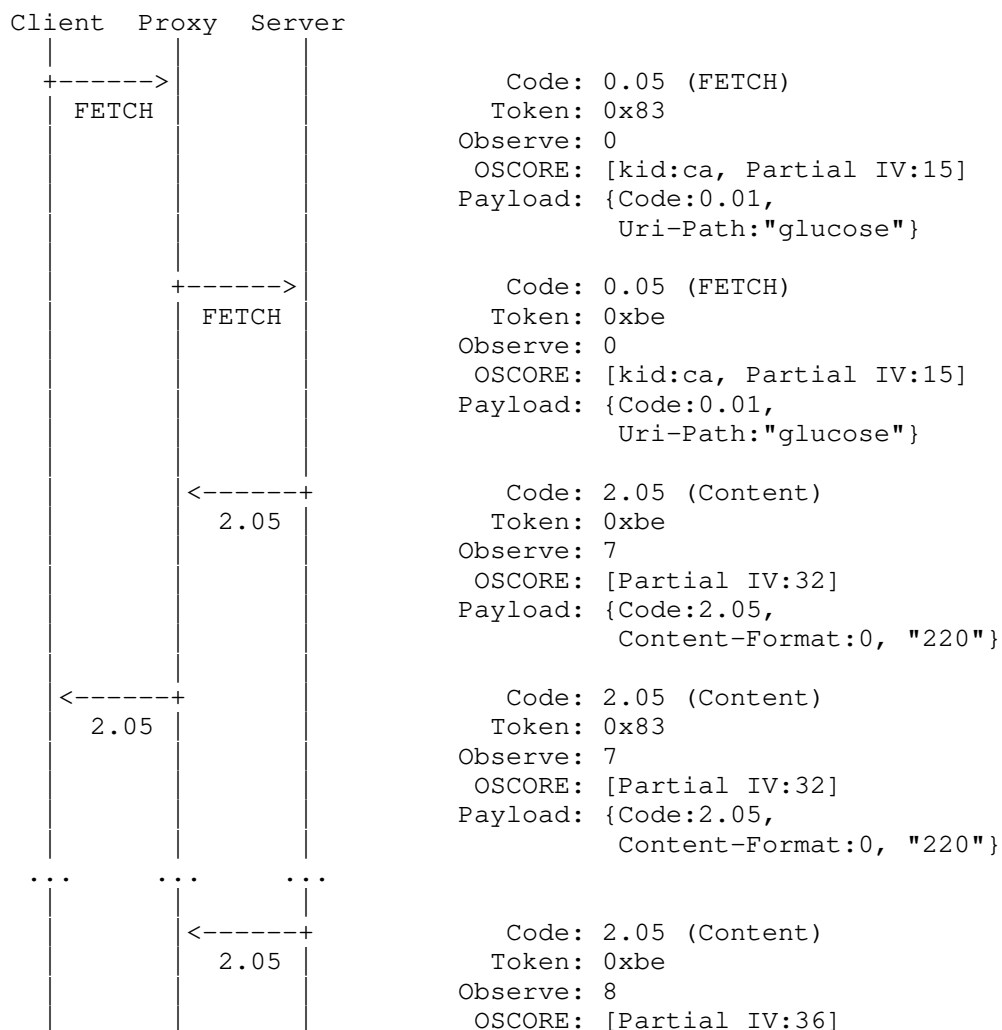
The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm\_status") and payload ("0") are encrypted.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a Partial IV (42).

The server verifies the request as specified in Section 8.2. The client verifies the response as specified in Section 8.4.

#### A.2. Secure Subscribe to Sensor

This example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), first receiving the value 220 mg/dl and then a second value 180 mg/dl.





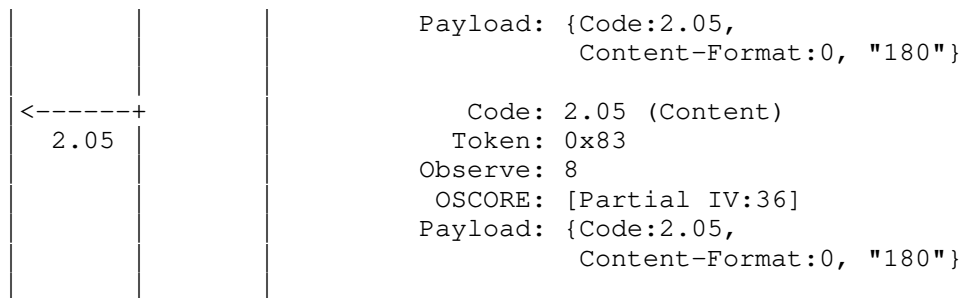


Figure 13: Secure Subscribe to Sensor. Square brackets [ ... ] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

The dummy Codes (FETCH/Content) are used to allow forwarding of Observe messages. The options Content-Format (0) and the payload ("220" and "180"), are encrypted.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Partial IV (15). The COSE headers of the responses contains Partial IVs (32 and 36).

The server verifies that the Partial IV has not been received before. The client verifies that the responses are bound to the request and that the Partial IVs are greater than any Partial IV previously received in a response bound to the request.

## Appendix B. Deployment Examples

For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device throughout its lifetime. Two examples are given in this section. In the first example, the security context is only derived once from the Master Secret. In the second example, security contexts are derived multiple times using random inputs.

### B.1. Security Context Derived Once

An application that only derives the security context once needs to handle the loss of mutable security context parameters, e.g. due to reboot.

#### B.1.1.1. Sender Sequence Number

In order to handle loss of Sender Sequence Numbers, the device may implement procedures for writing to non-volatile memory during normal operations and updating the security context after reboot, provided that the procedures comply with the requirements on the security context parameters (Section 3.3). This section gives an example of such a procedure.

There are known issues related to writing to non-volatile memory. For example, flash drives may have a limited number of erase operations during its life time. Also, the time for a write operation to non-volatile memory to be completed may be unpredictable, e.g. due to caching, which could result in important security context data not being stored at the time when the device reboots.

However, many devices have predictable limits for writing to non-volatile memory, are physically limited to only send a small amount of messages per minute, and may have no good source of randomness.

To prevent reuse of Sender Sequence Numbers (SSN), an endpoint may perform the following procedure during normal operations:

- o Before using a Sender Sequence Number that is evenly divisible by  $K$ , where  $K$  is a positive integer, store the Sender Sequence Number (SSN1) in non-volatile memory. After boot, the endpoint initiates the new Sender Sequence Number (SSN2) to the value stored in persistent memory plus  $K$  plus  $F$ :  $SSN2 = SSN1 + K + F$ , where  $F$  is a positive integer.
  - \* Writing to non-volatile memory can be costly; the value  $K$  gives a trade-off between frequency of storage operations and efficient use of Sender Sequence Numbers.
  - \* Writing to non-volatile memory may be subject to delays, or failure;  $F$  MUST be set so that the last Sender Sequence Number used before reboot is never larger than SSN2.

If  $F$  cannot be set so SSN2 is always larger than the last Sender Sequence Number used before reboot, the method described in this section MUST NOT be used.

#### B.1.1.2. Replay Window

In case of loss of security context on the server, to prevent accepting replay of previously received requests, the server may perform the following procedure after boot:

- o The server updates its Sender Sequence Number as specified in Appendix B.1.1, to be used as Partial IV in the response containing the Echo option (next bullet).
- o For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with an OSCORE protected 4.01 (Unauthorized), containing only the Echo option [I-D.ietf-core-echo-request-tag] and no diagnostic payload. The server MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response (see Section 5). If the server with use of the Echo option can verify a second OSCORE request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window of that security context.

#### B.1.3. Notifications

To prevent accepting replay of previously received notifications, the client may perform the following procedure after boot:

- o The client forgets about earlier registrations, removes all Notification Numbers and registers using Observe.

#### B.2. Security Context Derived Multiple Times

An application which does not require forward secrecy may allow multiple security contexts to be derived from one Master Secret. The requirements on the security context parameters MUST be fulfilled (Section 3.3) even if the client or server is rebooted, recommissioned or in error cases.

This section gives an example of a protocol which adds randomness to the ID Context parameter and uses that together with input parameters pre-established between client and server, in particular Master Secret, Master Salt, and Sender/Recipient ID (see Section 3.2), to derive new security contexts. The random input is transported between client and server in the 'kid context' parameter. This protocol MUST NOT be used unless both endpoints have good sources of randomness.

During normal requests the ID Context of an established security context may be sent in the 'kid context' which, together with 'kid', facilitates for the server to locate a security context. Alternatively, the 'kid context' may be omitted since the ID Context is expected to be known to both client and server, see Section 5.1.

The protocol described in this section may only be needed when the mutable part of security context is lost in the client or server,

e.g. when the endpoint has rebooted. The protocol may additionally be used whenever the client and server need to derive a new security context. For example, if a device is provisioned with one fixed set of input parameters (including Master Secret, Sender and Recipient Identifiers) then a randomized ID Context ensures that the security context is different for each deployment.

The protocol is described below with reference to Figure 14. The client or the server may initiate the protocol, in the latter case step 1 is omitted.

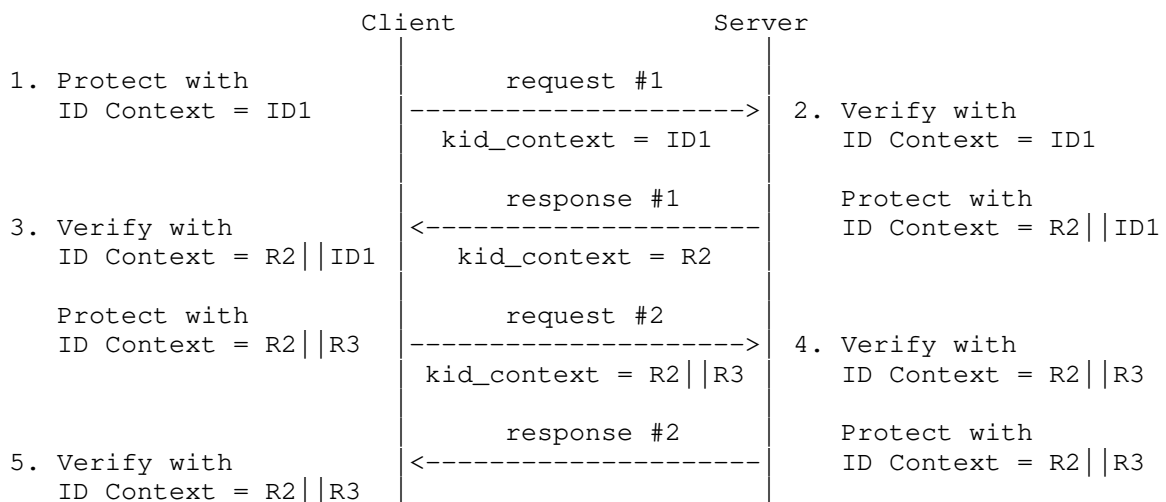


Figure 14: Protocol for establishing a new security context.

1. (Optional) If the client does not have a valid security context with the server, e.g. because of reboot or because this is the first time it contacts the server, then it generates a random string R1, and uses this as ID Context together with the input parameters shared with the server to derive a first security context. The client sends an OSCORE request to the server protected with the first security context, containing R1 wrapped in a CBOR bstr as 'kid context'. The request may target a special resource used for updating security contexts.
2. The server receives an OSCORE request for which it does not have a valid security context, either because the client has generated a new security context ID1 = R1, or because the server has lost part of its security context, e.g. ID Context, Sender Sequence Number or replay window. If the server is able to verify the request (see Section 8.2) with the new derived first security context using the received ID1 (transported in 'kid context') as

ID Context and the input parameters associated to the received 'kid', then the server generates a random string R2, and derives a second security context with ID Context = ID2 = R2 || ID1. The server sends a 4.01 (Unauthorized) response protected with the second security context, containing R2 wrapped in a CBOR bstr as 'kid context', and caches R2. R2 MUST NOT be reused as that may lead to reuse of key and nonce in response #1. Note that the server may receive several requests #1 associated with one security context, leading to multiple parallel protocol runs. Multiple instances of R2 may need to be cached until one of the protocol runs is completed, see Appendix B.2.1.

3. The client receives a response with 'kid context' containing a CBOR bstr wrapping R2 to an OSCORE request it made with ID Context = ID1. The client derives a second security context using ID Context = ID2 = R2 || ID1. If the client can verify the response (see Section 8.4) using the second security context, then the client makes a request protected with a third security context derived from ID Context = ID3 = R2 || R3, where R3 is a random byte string generated by the client. The request includes R2 || R3 wrapped in a CBOR bstr as 'kid context'.
4. If the server receives a request with 'kid context' containing a CBOR bstr wrapping ID3, where the first part of ID3 is identical to an R2 sent in a previous response #1 which it has not received before, then the server derives a third security context with ID Context = ID3. The server MUST NOT accept replayed request #2 messages. If the server can verify the request (see Section 8.2) with the third security context, then the server marks the third security context to be used with this client and removes all instances of R2 associated to this security context from the cache. This security context replaces the previous security context with the client, and the first and the second security contexts are deleted. The server responds using the same security context as in the request.
5. If the client receives a response to the request with the third security context and the response verifies (see Section 8.4), then the client marks the third security context to be used with this server. This security context replaces the previous security context with the server, and the first and second security contexts are deleted.

If verification fails in any step, the endpoint stops processing that message.

The length of the nonces R1, R2, and R3 is application specific. The application needs to set the length of each nonce such the

probability of its value being repeated is negligible; typically, at least 8 bytes long. Since R2 may be generated as the result of a replayed request #1, the probability for collision of R2s is impacted by the birthday paradox. For example, setting the length of R2 to 8 bytes results in an average collision after  $2^{32}$  response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Request #2 can be an ordinary request. The server performs the action of the request and sends response #2 after having successfully completed the security context related operations in step 4. The client acts on response #2 after having successfully completed step 5.

When sending request #2, the client is assured that the Sender Key (derived with the random value R3) has never been used before. When receiving response #2, the client is assured that the response (protected with a key derived from the random value R3 and the Master Secret) was created by the server in response to request #2.

Similarly, when receiving request #2, the server is assured that the request (protected with a key derived from the random value R2 and the Master Secret) was created by the client in response to response #1. When sending response #2, the server is assured that the Sender Key (derived with the random value R2) has never been used before.

Implementation and denial-of-service considerations are made in Appendix B.2.1 and Appendix B.2.2.

#### B.2.1. Implementation Considerations

This section add some implemention considerations to the protocol described in the previous section.

The server may only have space for a few security contexts, or only be able to handle a few protocol runs in parallel. The server may legitimately receive multiple request #1 messages using the same non-mutable security context, e.g. due to packet loss. replays of old request #1 messages could be difficult for the server to distinguish from legitimate. The server needs to handle the case when the maximum number of cached R2s is reached. If the server receives a request #1 and is not capable of executing it then it may respond with an unprotected 5.03 (Service Unavailable). The server may clear up state from protocol runs which never complete, e.g. set a timer when caching R2, and remove R2 and the associated security contexts from the cache at timeout. Additionally, state information can be flushed at reboot.

As an alternative to caching R2, the server could generate R2 in such a way that it can be sent (in response #1) and verified (at reception of request #2) as the value of R2 it had generated. Such a procedure MUST NOT lead to the server accepting replayed request #2 messages. One construction described in the following is based on using a secret random HMAC key K\_HMAC per set of non-mutable security context parameters associated to a client. This construction allows the server to handle verification of R2 in response #2 at the cost of storing the K\_HMAC keys and a slightly larger message overhead in response #1. Steps below refer to modifications to Appendix B.2:

- o In step 2, R2 is generated in the following way. First, the server generates a random K\_HMAC (unless it already has one associated with the security context), then it sets  $R2 = S2 \parallel \text{HMAC}(K\_HMAC, S2)$  where S2 is a random byte string, and the HMAC is truncated to 8 bytes. K\_HMAC may have an expiration time, after which it is erased. Note that neither R2, S2 nor the derived first and second security contexts need to be cached.
- o In step 4, instead of verifying that R2 coincides with a cached value, the server looks up the associated K\_HMAC and verifies the truncated HMAC, and the processing continues accordingly depending on verification success or failure. K\_HMAC is used until a run of the protocol is completed (after verification of request #2), or until it expires (whatever comes first), after which K\_HMAC is erased. (The latter corresponds to removing the cached values of R2 in step 4 of Appendix B.2, and makes the server reject replays of request #2.)

The length of S2 is application specific and the probability for collision of S2s is impacted by the birthday paradox. For example, setting the length of S2 to 8 bytes results in an average collision after  $2^{32}$  response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Two endpoints sharing a security context may accidentally initiate two instances of the protocol at the same time, each in the role of client, e.g. after a power outage affecting both endpoints. Such a race condition could potentially lead to both protocols failing, and both endpoints repeatedly re-initiating the protocol without converging. Both endpoints can detect this situation and it can be handled in different ways. The requests could potentially be more spread out in time, for example by only initiating this protocol when the endpoint actually needs to make a request, potentially adding a random delay before requests immediately after reboot or if such parallel protocol runs are detected.

### B.2.2. Attack Considerations

An on-path attacker may inject a message causing the endpoint to process verification of the message. A message crafted without access to the Master Secret will fail to verify.

Replaying an old request with a value of 'kid\_context' which the server does not recognize could trigger the protocol. This causes the server to generate the first and second security context and send a response. But if the client did not expect a response it will be discarded. This may still result in a denial-of-service attack against the server e.g. because of not being able to manage the state associated with many parallel protocol runs, and it may prevent legitimate client requests. Implementation alternatives with less data caching per request #1 message are favorable in this respect, see Appendix B.2.1.

Replaying response #1 in response to some request other than request #1 will fail to verify, since response #1 is associated to request #1, through the dependencies of ID Contexts and the Partial IV of request #1 included in the external\_aad of response #1.

If request #2 has already been well received, then the server has a valid security context, so a replay of request #2 is handled by the normal replay protection mechanism. Similarly if response #2 has already been received, a replay of response #2 to some other request from the client will fail by the normal verification of binding of response to request.

### Appendix C. Test Vectors

This appendix includes the test vectors for different examples of CoAP messages using OSCORE. Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server.

Note that in Appendix C.4 and all following test vectors the Token and the Message ID of the OSCORE-protected CoAP messages are set to the same value of the unprotected CoAP message, to help the reader with comparisons.

[NOTE: the following examples use option number = 9 (TBD1 assigned by IANA). If that differs, the RFC editor is asked to update the test vectors with data provided by the authors. Please remove this paragraph before publication.]



### C.1. Test Vector 1: Key Derivation with Master Salt

In this test vector, a Master Salt of 8 bytes is used. The default values are used for AEAD Algorithm and HKDF.

#### C.1.1. Client

##### Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

##### Outputs:

- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Recipient Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)
- o recipient nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

#### C.1.2. Server

##### Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)

- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Recipient Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)
- o recipient nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)

## C.2. Test Vector 2: Key Derivation without Master Salt

In this test vector, the default values are used for AEAD Algorithm, HKDF, and Master Salt.

### C.2.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x00 (1 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)

- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Recipient Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)
- o recipient nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

#### C.2.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x00 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Recipient Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

- o recipient nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)

### C.3. Test Vector 3: Key Derivation with ID Context

In this test vector, a Master Salt of 8 bytes and a ID Context of 8 bytes are used. The default values are used for AEAD Algorithm and HKDF.

#### C.3.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Recipient Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Recipient Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o recipient nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)

## C.3.2. Server

## Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Recipient Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

## Outputs:

- o Sender Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Recipient Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)
- o recipient nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

## C.4. Test Vector 4: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.1. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44015d1f00003974396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x (0 byte)
- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o external\_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0914 (2 bytes)
- o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f63616c686f7374620914ff612f1092f1776f1c1668b3825e (35 bytes)

#### C.5. Test Vector 5: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.2. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x440171c30000b932396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

Sender Context:

- o Sender ID: 0x00 (1 bytes)
- o Sender Key: 0x321b26943253c7fffb6003b0b64d74041 (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x00 (1 byte)
- o external\_aad: 0x8501810a4100411440 (9 bytes)
- o AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0x321b26943253c7fffb6003b0b64d74041 (16 bytes)
- o nonce: 0xbf35ae297d2dace910c52e99ed (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x091400 (3 bytes)
- o ciphertext: 0x4ed339a5a379b0b8bc731ffffb0 (13 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x440271c30000b932396c6f63616c686f737463091400ff4ed339a5a379b0b8bc731ffffb0 (36 bytes)

### C.6. Test Vector 6: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request for an application that sets the ID Context and requires it to be sent in the request, so 'kid context' is present in the protected message. This test vector uses the security context derived in Appendix C.3. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44012f8eef9bbf7a396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

Sender Context:

- o Sender ID: 0x (0 bytes)
- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o kid context: 0x37cbf3210017a2d3 (8 bytes)
- o external\_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o nonce: 0x2ca58fb85ff1b81c0b7181b84a (13 bytes)



From the previous parameter, the following is derived:

- o OSCORE option value: 0x19140837cbf3210017a2d3 (11 bytes)
- o ciphertext: 0x72cd7273fd331ac45cffbe55c3 (13 bytes)

From there:

- o Protected CoAP request (OSCORE message):  
0x44022f8eef9bbf7a396c6f63616c686f73746b19140837cbf3210017a2d3ff  
72cd7273fd331ac45cffbe55c3 (44 bytes)

#### C.7. Test Vector 7: OSCORE Response, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid' nor a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:  
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o external\_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- o nonce: 0x4622d4dd6d944168eeffb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x (0 bytes)
- o ciphertext: 0xdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (22 bytes)

From there:

- o Protected CoAP response (OSCORE message):  
0x64445d1f0000397490ffdbaad1e9a7e7b2a813d3c31524378303cdafae119106  
(32 bytes)

#### C.8. Test Vector 8: OSCORE Response with Partial IV, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid', but contains a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:  
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eeffb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x00 (1 byte)
- o external\_aad: 0x8501810a40411440 (8 bytes)

- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0100 (2 bytes)
- o ciphertext: 0x4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (22 bytes)

From there:

- o Protected CoAP response (OSCORE message): 0x64445d1f00003974920100ff4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (34 bytes)

## Appendix D. Overview of Security Properties

### D.1. Threat Model

This section describes the threat model using the terms of [RFC3552].

It is assumed that the endpoints running OSCORE have not themselves been compromised. The attacker is assumed to have control of the CoAP channel over which the endpoints communicate, including intermediary nodes. The attacker is capable of launching any passive or active, on-path or off-path attacks; including eavesdropping, traffic analysis, spoofing, insertion, modification, deletion, delay, replay, man-in-the-middle, and denial-of-service attacks. This means that the attacker can read any CoAP message on the network and undetectably remove, change, or inject forged messages onto the wire.

OSCORE targets the protection of the CoAP request/response layer (Section 2 of [RFC7252]) between the endpoints, including the CoAP Payload, Code, Uri-Path/Uri-Query, and the other Class E option instances (Section 4.1).

OSCORE does not protect the CoAP messaging layer (Section 2 of [RFC7252]) or other lower layers involved in routing and transporting the CoAP requests and responses.

Additionally, OSCORE does not protect Class U option instances (Section 4.1), as these are used to support CoAP forward proxy operations (see Section 5.7.2 of [RFC7252]). The supported proxies

(forwarding, cross-protocol e.g. CoAP to CoAP-mappable protocols such as HTTP) must be able to change certain Class U options (by instruction from the Client), resulting in the CoAP request being redirected to the server. Changes caused by the proxy may result in the request not reaching the server or reaching the wrong server. For cross-protocol proxies, mappings are done on the Outer part of the message so these protocols are essentially used as transport. Manipulation of these options may thus impact whether the protected message reaches or does not reach the destination endpoint.

Attacks on unprotected CoAP message fields generally causes denial-of-service attacks which are out of scope of this document, more details are given in Appendix D.5.

Attacks against the CoAP request-response layer are in scope. OSCORE is intended to protect against eavesdropping, spoofing, insertion, modification, deletion, replay, and man-in-the middle attacks.

OSCORE is susceptible to traffic analysis as discussed later in Appendix D.

## D.2. Supporting Proxy Operations

CoAP is designed to work with intermediaries reading and/or changing CoAP message fields to perform supporting operations in constrained environments, e.g. forwarding and cross-protocol translations.

Securing CoAP on transport layer protects the entire message between the endpoints in which case CoAP proxy operations are not possible. In order to enable proxy operations, security on transport layer needs to be terminated at the proxy in which case the CoAP message in its entirety is unprotected in the proxy.

Requirements for CoAP end-to-end security are specified in [I-D.hartke-core-e2e-security-reqs], in particular forwarding is detailed in Section 2.2.1. The client and server are assumed to be honest, while proxies and gateways are only trusted to perform their intended operations.

By working at the CoAP layer, OSCORE enables different CoAP message fields to be protected differently, which allows message fields required for proxy operations to be available to the proxy while message fields intended for the other endpoint remain protected. In the remainder of this section we analyze how OSCORE protects the protected message fields and the consequences of message fields intended for proxy operation being unprotected.

### D.3. Protected Message Fields

Protected message fields are included in the Plaintext (Section 5.3) and the Additional Authenticated Data (Section 5.4) of the COSE\_Encrypt0 object and encrypted using an AEAD algorithm.

OSCORE depends on a pre-established random Master Secret (Section 12.3) used to derive encryption keys, and a construction for making (key, nonce) pairs unique (Appendix D.4). Assuming this is true, and the keys are used for no more data than indicated in Section 7.2.1, OSCORE should provide the following guarantees:

- o Confidentiality: An attacker should not be able to determine the plaintext contents of a given OSCORE message or determine that different plaintexts are related (Section 5.3).
- o Integrity: An attacker should not be able to craft a new OSCORE message with protected message fields different from an existing OSCORE message which will be accepted by the receiver.
- o Request-response binding: An attacker should not be able to make a client match a response to the wrong request.
- o Non-replayability: An attacker should not be able to cause the receiver to accept a message which it has previously received and accepted.

In the above, the attacker is anyone except the endpoints, e.g. a compromised intermediary. Informally, OSCORE provides these properties by AEAD-protecting the plaintext with a strong key and uniqueness of (key, nonce) pairs. AEAD encryption [RFC5116] provides confidentiality and integrity for the data. Response-request binding is provided by including the 'kid' and Partial IV of the request in the AAD of the response. Non-replayability of requests and notifications is provided by using unique (key, nonce) pairs and a replay protection mechanism (application dependent, see Section 7.4).

OSCORE is susceptible to a variety of traffic analysis attacks based on observing the length and timing of encrypted packets. OSCORE does not provide any specific defenses against this form of attack but the application may use a padding mechanism to prevent an attacker from directly determine the length of the padding. However, information about padding may still be revealed by side-channel attacks observing differences in timing.

#### D.4. Uniqueness of (key, nonce)

In this section we show that (key, nonce) pairs are unique as long as the requirements in Sections 3.3 and 7.2.1 are followed.

Fix a Common Context (Section 3.1) and an endpoint, called the encrypting endpoint. An endpoint may alternate between client and server roles, but each endpoint always encrypts with the Sender Key of its Sender Context. Sender Keys are (stochastically) unique since they are derived with HKDF using unique Sender IDs, so messages encrypted by different endpoints use different keys. It remains to prove that the nonces used by the fixed endpoint are unique.

Since the Common IV is fixed, the nonces are determined by a Partial IV (PIV) and the Sender ID of the endpoint generating that Partial IV (ID\_PIV). The nonce construction (Section 5.2) with the size of the ID\_PIV (S) creates unique nonces for different (ID\_PIV, PIV) pairs. There are two cases:

A. For requests, and responses with Partial IV (e.g. Observe notifications):

- o ID\_PIV = Sender ID of the encrypting endpoint
- o PIV = current Partial IV of the encrypting endpoint

Since the encrypting endpoint steps the Partial IV for each use, the nonces used in case A are all unique as long as the number of encrypted messages is kept within the required range (Section 7.2.1).

B. For responses without Partial IV (e.g. single response to a request):

- o ID\_PIV = Sender ID of the endpoint generating the request
- o PIV = Partial IV of the request

Since the Sender IDs are unique, ID\_PIV is different from the Sender ID of the encrypting endpoint. Therefore, the nonces in case B are different compared to nonces in case A, where the encrypting endpoint generated the Partial IV. Since the Partial IV of the request is verified for replay (Section 7.4) associated to this Recipient Context, PIV is unique for this ID\_PIV, which makes all nonces in case B distinct.

## D.5. Unprotected Message Fields

This section analyses attacks on message fields which are not protected by OSCORE according to the threat model Appendix D.1.

### D.5.1. CoAP Header Fields

- o Version. The CoAP version [RFC7252] is not expected to be sensitive to disclosure. Currently there is only one CoAP version defined. A change of this parameter is potentially a denial-of-service attack. Future versions of CoAP need to analyze attacks to OSCORE protected messages due to an adversary changing the CoAP version.
- o Token/Token Length. The Token field is a client-local identifier for differentiating between concurrent requests [RFC7252]. CoAP proxies are allowed to read and change Token and Token Length between hops. An eavesdropper reading the Token can match requests to responses which can be used in traffic analysis. In particular this is true for notifications, where multiple responses are matched with one request. Modifications of Token and Token Length by an on-path attacker may become a denial-of-service attack, since it may prevent the client to identify to which request the response belongs or to find the correct information to verify integrity of the response.
- o Code. The Outer CoAP Code of an OSCORE message is POST or FETCH for requests with corresponding response codes. An endpoint receiving the message discards the Outer CoAP Code and uses the Inner CoAP Code instead (see Section 4.2). Hence, modifications from attackers to the Outer Code do not impact the receiving endpoint. However, changing the Outer Code from FETCH to a Code value for a method that does not work with Observe (such as POST) may, depending on proxy implementation since Observe is undefined for several Codes, cause the proxy to not forward notifications, which is a denial-of-service attack. The use of FETCH rather than POST reveals no more than what is revealed by the presence of the Outer Observe option.
- o Type/Message ID. The Type/Message ID fields [RFC7252] reveal information about the UDP transport binding, e.g. an eavesdropper reading the Type or Message ID gain information about how UDP messages are related to each other. CoAP proxies are allowed to change Type and Message ID. These message fields are not present in CoAP over TCP [RFC8323], and does not impact the request/response message. A change of these fields in a UDP hop is a denial-of-service attack. By sending an ACK, an attacker can make the endpoint believe that it does not need to retransmit the

previous message. By sending a RST, an attacker may be able to cancel an observation. By changing a NON to a CON, the attacker can cause the receiving endpoint to ACK messages for which no ACK was requested.

- o Length. This field contains the length of the message [RFC8323] which may be used for traffic analysis. These message fields are not present in CoAP over UDP, and does not impact the request/response message. A change of Length is a denial-of-service attack similar to changing TCP header fields.

#### D.5.2. CoAP Options

- o Max-Age. The Outer Max-Age is set to zero to avoid unnecessary caching of OSCORE error responses. Changing this value thus may cause unnecessary caching. No additional information is carried with this option.
- o Proxy-Uri/Proxy-Scheme. These options are used in CoAP forward proxy deployments. With OSCORE, the Proxy-Uri option does not contain the Uri-Path/Uri-Query parts of the URI. The other parts of Proxy-Uri cannot be protected because forward proxies need to change them in order to perform their functions. The server can verify what scheme is used in the last hop, but not what was requested by the client or what was used in previous hops.
- o Uri-Host/Uri-Port. In forward proxy deployments, the Uri-Host/Uri-Port may be changed by an adversary, and the application needs to handle the consequences of that (see Section 4.1.3.2). The Uri-Host may either be omitted, reveal information equivalent to that of the IP address or more privacy-sensitive information, which is discouraged.
- o Observe. The Outer Observe option is intended for a proxy to support forwarding of Observe messages, but is ignored by the endpoints since the Inner Observe determines the processing in the endpoints. Since the Partial IV provides absolute ordering of notifications it is not possible for an intermediary to spoof reordering (see Section 4.1.3.5). The absence of Partial IV, since only allowed for the first notification, does not prevent correct ordering of notifications. The size and distributions of notifications over time may reveal information about the content or nature of the notifications. Cancellations (Section 4.1.3.5.1) are not bound to the corresponding registrations in the same way responses are bound to requests in OSCORE (see Appendix D.3), but that does not open up for attacks based on mismatched cancellations, since for cancellations to be accepted, all options



in the decrypted message except for ETag Options MUST be the same (see Section 4.1.3.5).

- o Block1/Block2/Size1/Size2. The Outer Block options enables fragmentation of OSCORE messages in addition to segmentation performed by the Inner Block options. The presence of these options indicates a large message being sent and the message size can be estimated and used for traffic analysis. Manipulating these options is a potential denial-of-service attack, e.g. injection of alleged Block fragments. The specification of a maximum size of message, MAX\_UNFRAGMENTED\_SIZE (Section 4.1.3.4.2), above which messages will be dropped, is intended as one measure to mitigate this kind of attack.
- o No-Response. The Outer No-Response option is used to support proxy functionality, specifically to avoid error transmissions from proxies to clients, and to avoid bandwidth reduction to servers by proxies applying congestion control when not receiving responses. Modifying or introducing this option is a potential denial-of-service attack against the proxy operations, but since the option has an Inner value its use can be securely agreed between the endpoints. The presence of this option is not expected to reveal any sensitive information about the message exchange.
- o OSCORE. The OSCORE option contains information about the compressed COSE header. Changing this field may cause OSCORE verification to fail.

#### D.5.3. Error and Signaling Messages

Error messages occurring during CoAP processing are protected end-to-end. Error messages occurring during OSCORE processing are not always possible to protect, e.g. if the receiving endpoint cannot locate the right security context. For this setting, unprotected error messages are allowed as specified to prevent extensive retransmissions. Those error messages can be spoofed or manipulated, which is a potential denial-of-service attack.

This document specifies OPTIONAL error codes and specific diagnostic payloads for OSCORE processing error messages. Such messages might reveal information about how many and which security contexts exist on the server. Servers MAY want to omit the diagnostic payload of error messages, use the same error code for all errors, or avoid responding altogether in case of OSCORE processing errors, if that is a security concern for the application. Moreover, clients MUST NOT rely on the error code or the diagnostic payload to trigger specific

actions, as these errors are unprotected and can be spoofed or manipulated.

Signaling messages used in CoAP over TCP [RFC8323] are intended to be hop-by-hop; spoofing signaling messages can be used as a denial-of-service attack of a TCP connection.

#### D.5.4. HTTP Message Fields

In contrast to CoAP, where OSCORE does not protect header fields to enable CoAP-CoAP proxy operations, the use of OSCORE with HTTP is restricted to transporting a protected CoAP message over an HTTP hop. Any unprotected HTTP message fields may reveal information about the transport of the OSCORE message and enable various denial-of-service attacks. It is RECOMMENDED to additionally use TLS [RFC8446] for HTTP hops, which enables encryption and integrity protection of headers, but still leaves some information for traffic analysis.

#### Appendix E. CDDL Summary

Data structure definitions in the present specification employ the CDDL language for conciseness and precision. CDDL is defined in [I-D.ietf-cbor-cddl], which at the time of writing this appendix is in the process of completion. As the document is not yet available for a normative reference, the present appendix defines the small subset of CDDL that is being used in the present specification.

Within the subset being used here, a CDDL rule is of the form "name = type", where "name" is the name given to the "type". A "type" can be one of:

- o a reference to another named type, by giving its name. The predefined named types used in the present specification are: "uint", an unsigned integer (as represented in CBOR by major type 0); "int", an unsigned or negative integer (as represented in CBOR by major type 0 or 1); "bstr", a byte string (as represented in CBOR by major type 2); "tstr", a text string (as represented in CBOR by major type 3);
- o a choice between two types, by giving both types separated by a "/";
- o an array type (as represented in CBOR by major type 4), where the sequence of elements of the array is described by giving a sequence of entries separated by commas ",", and this sequence is enclosed by square brackets "[" and "]". Arrays described by an array description contain elements that correspond one-to-one to the sequence of entries given. Each entry of an array description

is of the form "name : type", where "name" is the name given to the entry and "type" is the type of the array element corresponding to this entry.

#### Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Ben Campbell, Esko Dijk, Jaro Fietz, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Mirja Kuehlewind, Kathleen Moriarty, Eric Rescorla, Michael Richardson, Adam Roach, Jim Schaad, Peter van der Stok, Dave Thaler, Martin Thomson, Marco Tiloca, William Vignat, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

#### Authors' Addresses

Goeran Selander  
Ericsson AB

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

John Mattsson  
Ericsson AB

Email: [john.mattsson@ericsson.com](mailto:john.mattsson@ericsson.com)

Francesca Palombini  
Ericsson AB

Email: [francesca.palombini@ericsson.com](mailto:francesca.palombini@ericsson.com)

Ludwig Seitz  
RISE SICS

Email: [ludwig.seitz@ri.se](mailto:ludwig.seitz@ri.se)

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 9, 2019

M. Tiloca  
RISE AB  
G. Selander  
F. Palombini  
Ericsson AB  
J. Park  
Universitaet Duisburg-Essen  
March 08, 2019

Group OSCORE - Secure Group Communication for CoAP  
draft-ietf-core-oscore-groupcomm-04

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 September 9, 2019.

Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	4
2. OSCORE Security Context . . . . .	5
2.1. Management of Group Keying Material . . . . .	8
2.2. Wrap-Around of Partial IVs . . . . .	9
3. The COSE Object . . . . .	9
3.1. Updated external_aad . . . . .	9
3.2. Use of the 'kid' Parameter . . . . .	10
3.3. Updated 'unprotected' Field . . . . .	10
4. OSCORE Header Compression . . . . .	10
4.1. Encoding of the OSCORE Option Value . . . . .	11
4.2. Encoding of the OSCORE Payload . . . . .	12
4.3. Examples of Compressed COSE Objects . . . . .	12
5. Message Binding, Sequence Numbers, Freshness and Replay Protection . . . . .	13
5.1. Synchronization of Sender Sequence Numbers . . . . .	13
6. Message Processing . . . . .	14
6.1. Protecting the Request . . . . .	14
6.2. Verifying the Request . . . . .	15
6.3. Protecting the Response . . . . .	15
6.4. Verifying the Response . . . . .	16
7. Responsibilities of the Group Manager . . . . .	17
8. Security Considerations . . . . .	18
8.1. Group-level Security . . . . .	18
8.2. Uniqueness of (key, nonce) . . . . .	18
8.3. Management of Group Keying Material . . . . .	19
8.4. Update of Security Context and Key Rotation . . . . .	19
8.5. Collision of Group Identifiers . . . . .	20
9. IANA Considerations . . . . .	20
9.1. Counter Signature Parameters Registry . . . . .	20
9.2. Expert Review Instructions . . . . .	22
10. References . . . . .	23
10.1. Normative References . . . . .	23
10.2. Informative References . . . . .	23
Appendix A. Assumptions and Security Objectives . . . . .	25
A.1. Assumptions . . . . .	25

A.2. Security Objectives . . . . .	26
Appendix B. List of Use Cases . . . . .	27
Appendix C. Example of Group Identifier Format . . . . .	29
Appendix D. Set-up of New Endpoints . . . . .	30
Appendix E. Examples of Synchronization Approaches . . . . .	31
E.1. Best-Effort Synchronization . . . . .	31
E.2. Baseline Synchronization . . . . .	31
E.3. Challenge-Response Synchronization . . . . .	32
Appendix F. No Verification of Signatures . . . . .	33
Appendix G. Document Updates . . . . .	34
G.1. Version -03 to -04 . . . . .	34
G.2. Version -02 to -03 . . . . .	35
G.3. Version -01 to -02 . . . . .	36
G.4. Version -00 to -01 . . . . .	36
Acknowledgments . . . . .	37
Authors' Addresses . . . . .	37

## 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] 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) [I-D.ietf-core-object-security] 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 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]; 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 [I-D.ietf-core-object-security].

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 should be unique within the set of groups of a given Group Manager, in order to avoid collisions. In case they are not, the considerations in Section 8.5 apply.
- 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 [I-D.ietf-core-object-security], 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.
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 [I-D.ietf-core-object-security], 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 [I-D.ietf-core-object-security], 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 the 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 [I-D.ietf-core-object-security], each Recipient Context stores also the public key of the associated other endpoint from which messages are received.

The table in Figure 1 overviews the new information included in the OSCORE Security Context, with respect to what defined in Section 3 of [I-D.ietf-core-object-security].

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.

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 [I-D.ietf-core-object-security]. 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 [I-D.ietf-core-object-security]. The mandatory-to-implement HKDF and AEAD algorithms for Group OSCORE are the same as in [I-D.ietf-core-object-security].

## 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. 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

A client can eventually experience a wrap-around of its own Sender Sequence Number, which is used as Partial IV in outgoing requests and incremented after each request.

When this happens, the endpoint **MUST NOT** transmit further group requests 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 [I-D.ietf-core-object-security].

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 [I-D.ietf-core-object-security], 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 Additional Data (AEAD) algorithm. For Group OSCORE, the following modifications apply.

### 3.1. Updated external\_aad

The external\_aad in the Additional Authenticated Data (AAD) is extended with the counter signature algorithm and related parameters used to sign messages. In particular, compared with Section 5.4 of [I-D.ietf-core-object-security], 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. Note that if the Counter Signature Parameters in the Common Context is empty, 'par\_countersign' is not present.

This external\_aad structure is used both for the encryption process producing the ciphertext (see Section 5.3 of [RFC8152]) and 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],  
  request_kid : bstr,  
  request_piv : bstr,  
  options : bstr  
]
```

### 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 [I-D.ietf-core-object-security], 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 above and the ciphertext of the COSE\_Encrypt0 object as payload.

## 4. OSCORE Header Compression

The OSCORE compression defined in Section 6 of [I-D.ietf-core-object-security] 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 [I-D.ietf-core-object-security], 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 [I-D.ietf-core-object-security], 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 [I-D.ietf-core-object-security].

- 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.

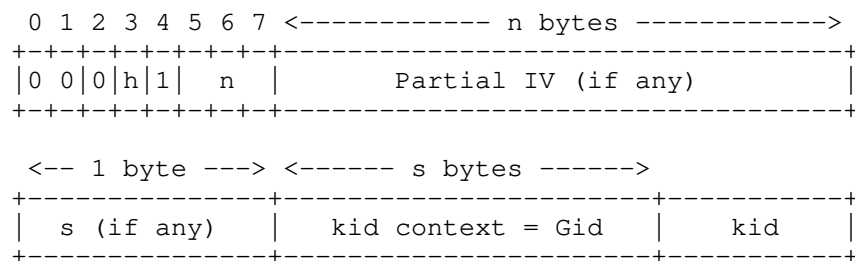


Figure 2: OSCORE Option Value

#### 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 [I-D.ietf-core-object-security] 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 [I-D.ietf-core-object-security] 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.

Besides, group OSCORE provides additional assurances on the client side, upon receiving responses bound to a same request. That is, as long as the client retains the CoAP Token used in a request (see Section 2.5 of [RFC7390]), group OSCORE ensures that: any possible response sent to that request is not a replay; and at most one response to that request from a given server is accepted, if required by the application.

More details about error processing for replay detection in group OSCORE are specified in Section 6 of this specification. The mechanisms describing replay protection and freshness of Observe notifications do not apply to group OSCORE, as Observe is not defined for group settings.

### 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 [I-D.ietf-core-object-security], 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, message integrity.

As per [RFC7252][RFC7390], 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 [I-D.ietf-core-object-security]. 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 [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 4, the encryption of the COSE object is modified as described in Section 3. The encoding of the compressed COSE object is modified as described in Section 4.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2. 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 [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the decoding of the compressed COSE object follows Section 4. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the server creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], also retrieving the client's public key.
- o In step 4, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 6, the server also verifies the counter signature using the public key of the client from the associated Recipient Context.
- 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 [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.

- o In step 4, the encryption of the COSE object is modified as described in Section 3. The encoding of the compressed COSE object is modified as described in Section 4.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2. 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 [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the decoding of the compressed COSE object is modified as described in Section 3. The client also checks whether it previously received a secure response to this request, such that it was successfully verified and included the same Recipient ID ('kid') of the just received response. In case of positive match the client SHALL stop processing the response. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the client creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], also retrieving the server's public key.
- o In step 3, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 5, the client also verifies the counter signature using the public key of the server from the associated Recipient Context. In case of success, the client also records the received Recipient ID ('kid') as included in a successfully verified response to the request.
- 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.

Upon freeing up the Token value of a secure group request for possible reuse [RFC7390], the client MUST delete the list of recorded Recipient IDs associated to that request (see step 5 above).

## 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. Such policies can be enforced locally by the Group Manager, or by a third party in a trust relation with the Group Manager and entrusted to enforce join policies on behalf of the Group Manager.
3. Driving the join process to add new endpoints as group members.
4. Establishing Security Common Contexts and providing them to authorized group members during the join process, together with a corresponding Security 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.

## 8. Security Considerations

The same security considerations from OSCORE (Section 11 of [I-D.ietf-core-object-security]) apply to this specification. Additional security aspects to be taken into account are discussed below.

### 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.3 of [I-D.ietf-core-object-security] 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.3 of [I-D.ietf-core-object-security] for messages including a Partial IV concerns only group requests, and same considerations from [I-D.ietf-core-object-security] apply here as well.

- o The case B in Appendix D.3 of [I-D.ietf-core-object-security] for messages not including a Partial IV concerns all group responses, and same considerations from [I-D.ietf-core-object-security] 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. Note that a former (compromised) group member can take advantage of this by sending messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the storage 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. That can also happen if the application can not guarantee unique Group Identifiers within a given Group Manager. 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, and as long as the Sender IDs within a group are unique, AEAD keys are different among different groups.

#### 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 9.2.

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]
RSAES-OAEP w/ RFC 8017 default parameters	-40		Parameters not present	[This Document]
RSAES-OAEP w/ SHA-256	-41		Parameters not present	[This Document]
RSAES-OAEP w/ SHA-512	-42		Parameters not present	[This Document]

## 9.2. Expert Review Instructions

The IANA Registry established in this document is 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:

TBD

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-object-security]  
Selandier, 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.
- [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>>.
- [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>>.

### 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-00 (work in progress), December 2018.
- [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-22 (work in progress), March 2019.
- [I-D.ietf-core-echo-request-tag]  
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [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 [I-D.ietf-core-object-security], 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] 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] 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 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 LLNs 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 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.

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 impacts 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 Security 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 OSCORE 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 OSCORE 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 Section 7.5.2 of [I-D.ietf-core-object-security].

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 a 4.03 Forbidden response message including an Echo Option, and stores the option value included therein.

Upon receiving a 4.03 Forbidden 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. In 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 4.03 Forbidden 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 -03 to -04

- o Added the new "Counter Signature Parameters" in the Security 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.2. 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.3. 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.4. 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, 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](mailto:goran.selander@ericsson.com)

Francesca Palombini  
Ericsson AB  
Torshamnsgatan 23  
Kista SE-16440 Stockholm  
Sweden

Email: [francesca.palombini@ericsson.com](mailto:francesca.palombini@ericsson.com)

Jiye Park  
Universitaet Duisburg-Essen  
Schuetzenbahn 70  
Essen 45127  
Germany

Email: [ji-ye.park@uni-due.de](mailto:ji-ye.park@uni-due.de)

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

K. Lynn  
Oracle+Dyn  
P. van der Stok  
Consultant  
M. Koster  
SmartThings  
C. Amsuess  
Energy Harvesting Solutions  
March 11, 2019

CoRE Resource Directory: DNS-SD mapping  
draft-ietf-core-rd-dns-sd-04

Abstract

Resource and service discovery are complementary. Resource discovery provides fine-grained detail about the content of a web server, while service discovery can provide a scalable method to locate servers in large networks. This document defines a method for mapping between CoRE Link Format attributes and DNS-Based Service Discovery records to facilitate the use of either method to locate RESTful service interfaces (APIs) in heterogeneous HTTP/CoAP 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 September 12, 2019.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction and Background . . . . .	2
1.1. Terminology . . . . .	3
1.2. CoRE Resource Discovery . . . . .	4
1.3. CoRE Resource Directories . . . . .	5
1.4. DNS-Based Service Discovery . . . . .	5
2. New Link-Format Attributes . . . . .	6
2.1. Export attribute "exp" . . . . .	7
2.2. Resource Instance attribute "ins=" . . . . .	7
2.3. Service Type attribute "st=" . . . . .	7
3. Mapping CoRE Link Attributes to DNS-SD Record Fields . . . . .	7
3.1. Mapping Resource Instance attribute "ins=" to <Instance> . . . . .	7
3.2. Mapping Service Type attribute "st=" to <ServiceType> . . . . .	8
3.3. <Domain> Mapping . . . . .	8
3.4. TXT Record key=value strings . . . . .	9
3.5. Exporting resource links into DNS-SD . . . . .	9
4. IANA considerations . . . . .	10
5. Security considerations . . . . .	10
6. References . . . . .	10
6.1. Normative References . . . . .	10
6.2. Informative References . . . . .	12
Appendix A. Acknowledgments . . . . .	12
Authors' Addresses . . . . .	13

## 1. Introduction and Background

The Constrained RESTful Environments (CoRE) working group aims at realizing the [REST] architecture in a suitable form for the most constrained devices (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN [RFC4944]). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation. The main deliverable of CoRE is the Constrained Application Protocol (CoAP) specification [RFC7252].

CoRE Link Format [RFC6690] is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other related resources. Automated dynamic discovery of resources hosted by a constrained server is critical in M2M applications, where human

intervention is minimal and static configurations result in brittleness.

DNS-Based Service Discovery (DNS-SD) [RFC6763] supports wide-area search for instances of a given service type (i.e. servers that support a particular application protocol stack). A service instance consists of a server's name, IP address, and port number plus additional meta-data about the server. This data may extend to support multi-function devices, where multiple services are available at the same endpoint. The result of the discovery process may include a path to a resource representing the entry point to each function's RESTful service interface and possibly a link to a formal description of that interface (e.g. a JSON Hyper-Schema document [I-D.handrews-json-schema-hyperschema]).

Resource and service discovery are complementary in the case of large networks, where the latter can facilitate scaling. This document defines a mapping between CoRE Link Format attributes and DNS-Based Service Discovery records that permits discovery of CoAP services by either method. It also addresses the CoRE charter goal to interoperate with DNS-SD.

The primary use case for mapping between resource and service discovery is to support heterogeneous HTTP/CoAP environments where, for example, HTTP clients may discover and communicate with CoAP servers that are behind a "cross proxy" [RFC8075].

### 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 [RFC2119]. The term "byte" is used in its now conventional sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC6690] and [RFC8288]. Readers should also be familiar with the terms and concepts discussed in [RFC7252].

This specification also incorporates the terminology of [I-D.ietf-core-resource-directory].

In particular, the following terms are used frequently:

Endpoint: a web server associated with a specific IP address and port; thus a physical device may host one or more endpoints. Endpoints may also act as clients.

Link: Web Linking [RFC8288] defines a Web Link (link) as a typed connection between two resources, comprised of:

- o a link context,
- o a link relation type (see Section 2.1 of [RFC8288],
- o a link target, and
- o optionally, target attributes (see Section 2.2 of [RFC8288]).

A link can be viewed as a statement of the form "link context has a link relation type resource at link target, which (optionally) has target attributes", where link target and context are typically Universal Resource Identifiers (URIs) [RFC3986]. For example, "https://www.example.com/" has a "canonical" resource at "https://example.com", which has a "type" of "text/html".

## 1.2. CoRE Resource Discovery

The main function of Resource Discovery is to return links to the resources hosted by a server, complemented by attributes about those resources and additional link relations. In CoRE this collection of links and attributes is itself a resource (in contrast to HTTP, where headers delivered with a specific resource describe its attributes).

Resource Discovery can be performed either unicast or multicast. When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS) [RFC1034][RFC1035], unicast discovery is performed in order to locate the entry point to the resource of interest. This is performed using a GET to `"/.well-known/core"` on the server, which returns a payload in the CoRE Link Format [RFC6690]. A client would then match the appropriate Resource Type, Interface Description, and possible media type [RFC2045] for its application. These attributes may also be included in the query string in order to filter the number of links returned in a response.

Multicast Resource Discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A GET request to the appropriate multicast address is made for `"/.well-known/core"`. In order to limit the number and size of responses, a query string is recommended with the known attributes. Typically, a resource would be discovered based on its Resource Type and/or Interface Description, along with possible application-specific attributes.

### 1.3. CoRE Resource Directories

In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, limited bandwidth, or networks where multicast traffic is inefficient. These problems can be solved by deploying a network element called a Resource Directory (RD), which hosts descriptions of resources that originate on other endpoints and allows indirect lookups to be performed for those resources.

The Resource Directory implements a set of REST interfaces for endpoints to register and maintain collections of links, called Resource Directory registrations. [I-D.ietf-core-resource-directory] specifies the web interfaces that an RD supports for endpoints to discover the RD and to register, maintain, lookup and remove resource descriptions; for the RD to validate entries; and for clients to lookup resources from the RD.

### 1.4. DNS-Based Service Discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional method of naming and configuring DNS PTR, SRV, and TXT resource records to facilitate discovery of services (such as CoAP servers in a subdomain) using the existing DNS infrastructure. This section gives a brief overview of DNS-SD; for a detailed specification see [RFC6763].

DNS-SD Service Names are limited to 255 bytes and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>

The Service Name identifies a SRV/TXT resource record (RR) pair. The SRV RR specifies the hostname and port of an endpoint. The TXT RR provides additional information in the form of key/value pairs. DNS-Based Service Discovery is accomplished by sending a DNS request for PTR records with the name <ServiceType>.<Domain>, which will return a list of zero or more Service Names.

The <Domain> part of the Service Name is identical to the global (DNS subdomain) part of the authority in URIs [RFC3986] that identify the resources on an individual server or group of servers.

The <ServiceType> part is generally composed of two labels. The first label of the pair is the application protocol name [RFC6335] preceded by an underscore character. For example, an organization such as the Open Connectivity Foundation [OCF] that specifies Resource Types [RFC6335] might register application protocol names beginning with "oic", which all servers that advertise OCF resources would use as part of their ServiceType. The second label indicates

the transport protocol binding and is typically "\_udp" for CoAP services.

The default <Instance> part of the Service Name SHOULD be set to a default value at the factory and MAY be modified during the commissioning process. It MUST uniquely identify an instance of <ServiceType> within a <Domain>. Taken together, these three elements comprise a unique name for an SRV/TXT record pair within the DNS subdomain.

The granularity of a Service Name MAY be that of a host or group, or it might represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the endpoint, while protocol is part of the Service Name. In the case where a Service Name identifies a particular resource, the path part of the URI must be carried in a corresponding TXT record.

A DNS TXT record is in practice limited to a few hundred bytes in length, which is indicated in the resource record header in the DNS response message (See section 6 of [RFC6763]). The data consist of one or more strings comprising a key/value pair. By convention, the first pair is txtver=<number> (to support different versions of a service description). Each string is formatted as a single length byte followed by 0-255 bytes of text. An example string is:

```
-----  
| 0x08 | t | x | t | v | e | r | = | 1 |  
-----
```

## 2. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is often useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690] to enable the data-driven mappings described in Section 3:

```
link-extension    = ( "exp" )  
link-extension    = ( "ins" "=" (ptoken | quoted-string) )  
                  ; The token or string is max 63 bytes  
link-extension    = ( "st" "=" (ptoken | quoted-string) )  
                  ; The token or string is max 15 bytes
```

### 2.1. Export attribute "exp"

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported from a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally; for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7 of [I-D.ietf-core-resource-directory].

### 2.2. Resource Instance attribute "ins="

The Resource Instance "ins=" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources in a Resource Directory. This attribute specifies the value to be used for the <Instance> portion of an exported DNS-SD Service Name (see Section 1.4), and SHOULD be unique across resources with the same Resource Type "rt=" attribute in the domain in which it is used.

A Resource Instance SHOULD be a descriptive human readable string like "Ceiling Light, Room 3". This attribute MUST NOT be more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7 of [I-D.ietf-core-resource-directory].

### 2.3. Service Type attribute "st="

The Service Type instance "st=" attribute specifies the value to be used for the <ServiceType> portion of an exported DNS-SD Service Name (see Section 1.4). This attribute MUST NOT be more than 15 bytes in length (see [RFC6335], Section 5.1) and MUST be present in the IANA Service Name registry [st].

## 3. Mapping CoRE Link Attributes to DNS-SD Record Fields

### 3.1. Mapping Resource Instance attribute "ins=" to <Instance>

The Resource Instance "ins=" attribute maps directly to the <Instance> part of a DNS-SD Service Name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198]



text. However, if the "ins=" attribute is chosen to match the DNS host name of a service, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all (see Appendix D of [RFC6763]).

DNS labels are limited to 63 bytes in length and the entire Service Name may not exceed 255 bytes.

### 3.2. Mapping Service Type attribute "st=" to <ServiceType>

The Service Type "st=" attribute maps directly to the <ServiceType> part of a DNS-SD Service Name.

In practice, the ServiceType should unambiguously identify interoperable devices. It is up to individual SDOs to specify how to represent their registered Resource Type "rt=" values as registered application protocol names according to [RFC6335]. The application name is then used as the value of the resource "st=" attribute.

The resulting application protocol name MUST be composed of at least a single Net-Unicode text string, without underscore '\_' or period '.' and limited to 15 bytes in length (see Section 5.1 of [RFC6335]). This string is mapped to the DNS-SD <ServiceType> by prepending an underscore and appending a period followed by the "\_udp" label. For example, rt="oic.d.light" might correspond to the registered application protocol name st="oic-d-light" and would be mapped into Service Type "\_oic-d-light.\_udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

### 3.3. <Domain> Mapping

TBD: A method must be specified to determine which DNS zone the CoAP service description should be exported to. See, for example, Section 11 in [RFC6763] and Section 2 in [I-D.sctl-service-registration].

### 3.4. TXT Record key=value strings

DNS-SD key/value pairs may be derived from CoRE Link Format information and exported as key=value strings in a DNS-SD TXT record (See Section 6.3 of [RFC6763]).

The resource <URI> is exported as key/value pair "path=<URI>".

The Interface Description "if=" attribute is exported as key/value pair "if=<Interface Description>".

The DNS TXT record can be further populated by importing any other resource description attributes as they share the same key=value format specified in Section 6 of [RFC6763].

### 3.5. Exporting resource links into DNS-SD

Assuming the ability to query a Resource Directory or multicast a GET (?exp) over the local link, CoAP resource discovery may be used to populate the DNS-SD database in an automated fashion. CoAP resource descriptions (links) can be exported to DNS-SD for exposure to service discovery by using the Resource Instance attribute as the basis for a unique Service Name, composed with the Service Type attribute as the <ServiceType>, and registered in the appropriate <Domain>. The agent responsible for exporting records to the DNS zone file SHOULD be authenticated to the DNS server. The following example, using the example lookup location /rd-lookup, shows an agent discovering a resource to be exported:

```
Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>;
  exp;st=oic-d-light;rt="oic.d.light";ins="Spot";
  d="office";ep="nodel"
```

The agent subsequently registers the following DNS-SD RRs, assuming a derived DNS zone name "office.example.com":

```
_oic-d-light._udp.office.example.com IN PTR
  Spot._oic-d-light._udp.office.example.com
Spot._oic-d-light._udp.office.example.com IN TXT
  txtver=1;path=/light/1;rt=oic.d.light
Spot._oic-d-light._udp.office.example.com IN SRV  0 0 5683
  nodel.office.example.com.
nodel.office.example.com. IN AAAA  FDFD::1234
```

#### 4. IANA considerations

This specification defines new parameters for the registry "RD Parameters" provided under "CoRE Parameters" (TBD).

Full name	Short	Validity	Use	Description
ServiceType	st		RLA	Name of the Service Type, max 63 bytes
Resource Instance	ins		RLA	Instance identifier of the resource
Export	exp		RLA	flag to indicate exportation

#### 5. Security considerations

TBD

#### 6. References

##### 6.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <<https://www.rfc-editor.org/info/rfc2045>>.
- [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>>.

- [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>>.
- [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>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<https://www.rfc-editor.org/info/rfc5198>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [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>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

## 6.2. Informative References

- [I-D.handrews-json-schema-hyperschema]  
Andrews, H. and A. Wright, "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON", draft-handrews-json-schema-hyperschema-01 (work in progress), January 2018.
- [I-D.ietf-core-resource-directory]  
Shelby, Z., Kostner, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-19 (work in progress), January 2019.
- [I-D.sctl-service-registration]  
Cheshire, S. and T. Lemon, "Service Registration Protocol for DNS-Based Service Discovery", draft-sctl-service-registration-02 (work in progress), July 2018.
- [OCF]  
Open Connectivity Foundation, "OCF Specification 2.0", 2018, <<https://openconnectivity.org/developer/specifications>>.
- [REST]  
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [rt]  
IANA, "Resource Type (rt=) Link Target Attribute Values", 2012, <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#rt-link-target-att-value>>.
- [st]  
IANA, "Service Name and Transport Protocol Port Number Registry", 2018, <<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml>>.

## Appendix A. Acknowledgments

This document was split out from [I-D.ietf-core-resource-directory]. Zach Shelby was a co-author of the original version of this draft.

The authors wish to thank Stuart Cheshire, Ted Lemon, and David Thaler for their thorough reviews and clarifying suggestions.

Authors' Addresses

Kerry Lynn  
Oracle+Dyn  
150 Dow Street, Tower Two  
Manchester, NH 03101  
USA

Phone: +1 978-460-4253  
Email: kerlyn@ieee.org

Peter van der Stok  
Consultant

Phone: +31 492474673 (Netherlands), +33 966015248 (France)  
Email: consultancy@vanderstok.org  
URI:   www.vanderstok.org

Michael Koster  
SmartThings  
665 Clyde Avenue  
Mountain View, CA 94043  
USA

Phone: +1 707-502-5136  
Email: Michael.Koster@smartthings.com

Christian Amsuess  
Energy Harvesting Solutions  
Hollandstr. 12/4  
1020  
Austria

Phone: +43 664-9790639  
Email: c.amsuess@energyharvesting.at

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

Z. Shelby  
ARM  
M. Koster  
SmartThings  
C. Bormann  
Universitaet Bremen TZI  
P. van der Stok  
consultant  
C. Amsuess, Ed.  
March 11, 2019

CoRE Resource Directory  
draft-ietf-core-resource-directory-20

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 September 12, 2019.

## Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	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 . . . . .	13
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 . . . . .	28
5.3.1. Registration Update . . . . .	29
5.3.2. Registration Removal . . . . .	32
5.3.3. Further operations . . . . .	32
6. RD Lookup . . . . .	33
6.1. Resource lookup . . . . .	33
6.2. Lookup filtering . . . . .	34
6.3. Resource lookup examples . . . . .	36
6.4. Endpoint lookup . . . . .	38
7. Security policies . . . . .	39
7.1. Secure RD discovery . . . . .	40
7.2. Secure RD filtering . . . . .	40
7.3. Secure endpoint Name assignment . . . . .	41



8.	Security Considerations . . . . .	41
8.1.	Endpoint Identification and Authentication . . . . .	41
8.2.	Access Control . . . . .	42
8.3.	Denial of Service Attacks . . . . .	42
9.	IANA Considerations . . . . .	42
9.1.	Resource Types . . . . .	43
9.2.	IPv6 ND Resource Directory Address Option . . . . .	43
9.3.	RD Parameter Registry . . . . .	43
9.3.1.	Full description of the "Endpoint Type" Registration Parameter . . . . .	45
9.4.	"Endpoint Type" (et=) RD Parameter values . . . . .	45
9.5.	Multicast Address Registration . . . . .	46
10.	Examples . . . . .	46
10.1.	Lighting Installation . . . . .	46
10.1.1.	Installation Characteristics . . . . .	47
10.1.2.	RD entries . . . . .	48
10.2.	OMA Lightweight M2M (LWM2M) Example . . . . .	50
10.2.1.	The LWM2M Object Model . . . . .	51
10.2.2.	LWM2M Register Endpoint . . . . .	52
10.2.3.	LWM2M Update Endpoint Registration . . . . .	54
10.2.4.	LWM2M De-Register Endpoint . . . . .	54
11.	Acknowledgments . . . . .	54
12.	Changelog . . . . .	54
13.	References . . . . .	63
13.1.	Normative References . . . . .	63
13.2.	Informative References . . . . .	64
Appendix A.	Groups Registration and Lookup . . . . .	66
Appendix B.	Web links and the Resource Directory . . . . .	67
B.1.	A simple example . . . . .	67
B.1.1.	Resolving the URIs . . . . .	68
B.1.2.	Interpreting attributes and relations . . . . .	68
B.2.	A slightly more complex example . . . . .	69
B.3.	Enter the Resource Directory . . . . .	69
B.4.	A note on differences between link-format and Link headers . . . . .	71
Appendix C.	Limited Link Format . . . . .	72
Authors' Addresses	. . . . .	72

## 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.

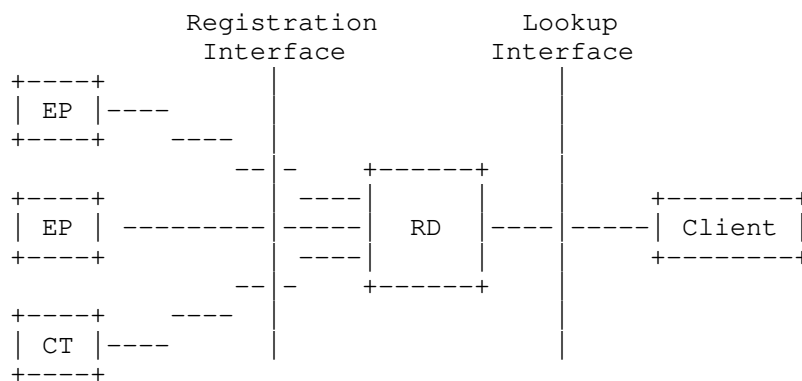


Figure 1: The resource directory architecture.

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.

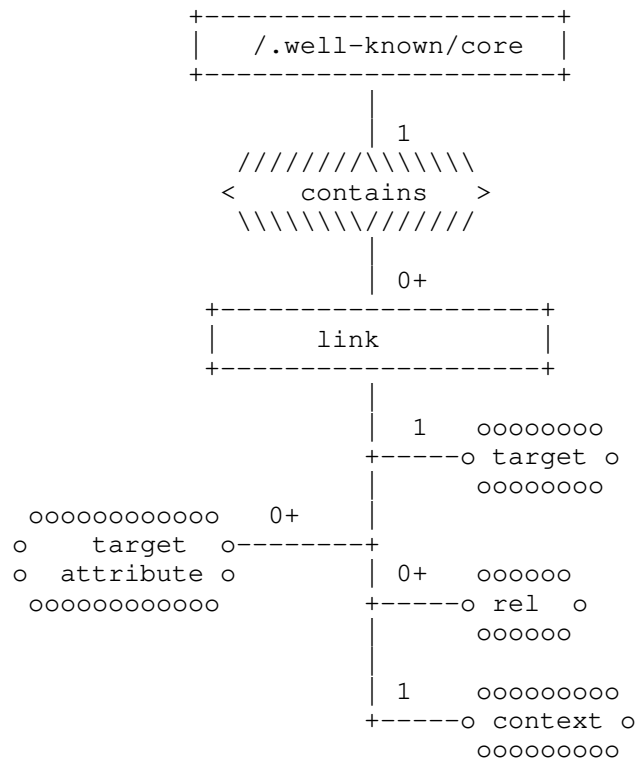


Figure 2: E-R Model of the content of `/.well-known/core`

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.



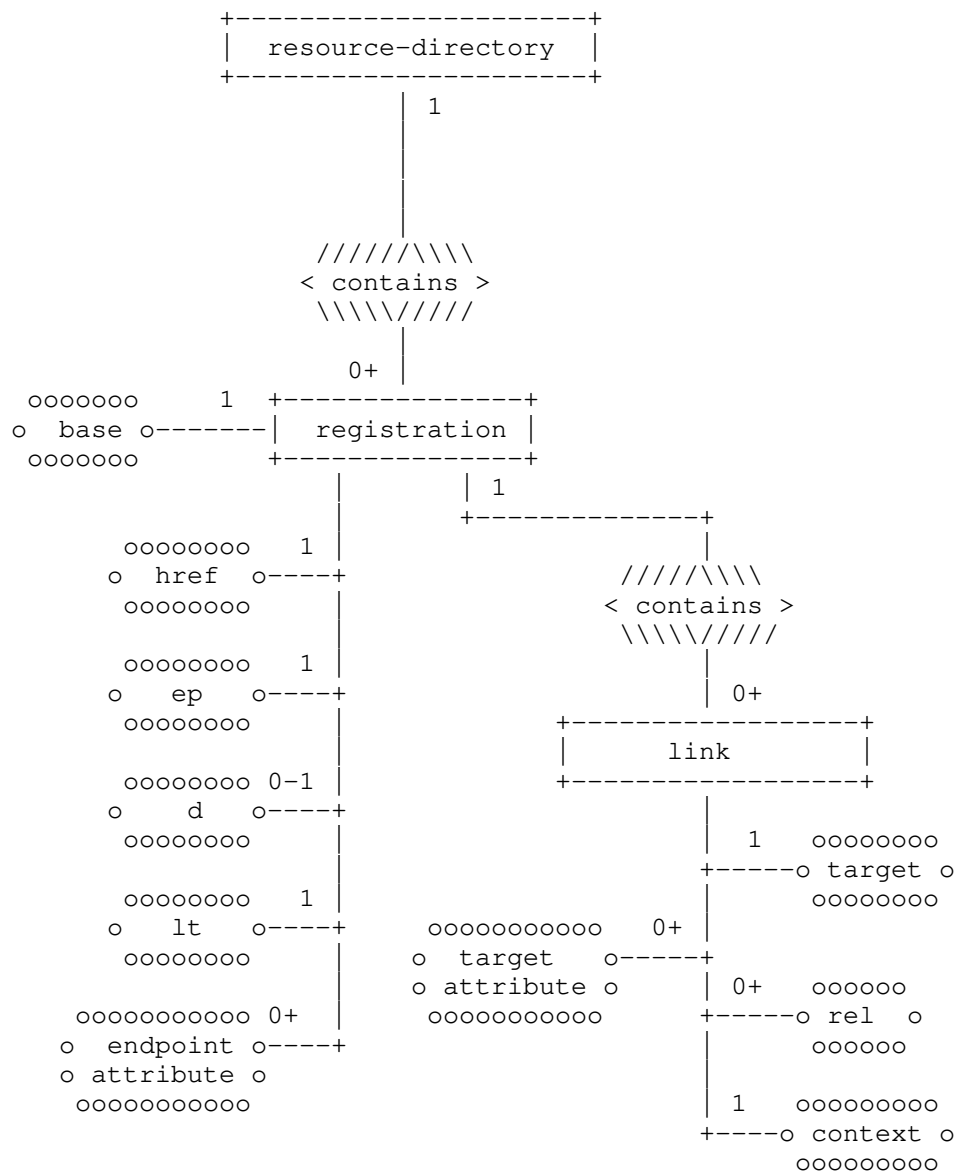


Figure 3: E-R Model of the content of the Resource Directory

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.

### 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.

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.
3. It may be configured to use a service discovery mechanism such as DNS-SD [RFC6763]. The present specification suggests configuring

the service with name `rd._sub._coap._udp`, preferably within the domain of the querying nodes.

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.

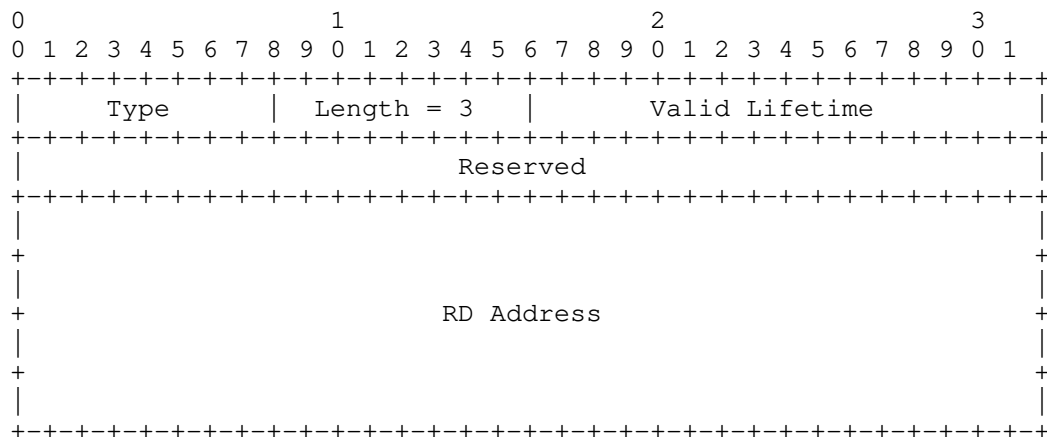
If multiple candidate addresses are discovered, the device may pick any of them initially, unless the discovery method indicates a more precise selection scheme.

#### 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.

[ The RFC editor is asked to replace this and later occurrences of MCD1 with the assigned IPv6 site-local address for "all CoRE Resource Directories". ]

```
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",
```

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"
```

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";
    anchor="coap://spurious.example.com:5683",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 6: 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=node1&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";
    anchor="coap://spurious.example.com:5683",
</sensors/light>;ct=41;rt="light-lux";if="sensor"

Res: 201 Created
Location: /rd/4521
```

### 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)
```

- 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>

With this response, the RD would answer the previous step's request:

Res: 2.04 Changed

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.

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

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 6

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"; anchor="coap://spurious.example.com:5683",  
<coap://local-proxy-old.example.com:5683/sensors/light>;ct=41;  
  rt="light-lux"; if="sensor";  
  anchor="coap://local-proxy-old.example.com:5683"
```

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

The consecutive query returns:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content

Payload:

```
<coaps://new.example.com:5684/sensors/temp>;ct=41;rt="temperature";  
  anchor="coap://spurious.example.com:5683",  
<coaps://new.example.com:5684/sensors/light>;ct=41;rt="light-lux";  
  if="sensor"; anchor="coaps://new.example.com:5684",
```

### 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

### 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 vice versa, 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"
```

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] "
```

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"
```

The following example shows a client performing a lookup of all resources from endpoints 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"
```

#### 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"
```

## 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 proof 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	60-4294967295	RLA	Name of the endpoint, max 63 bytes
Lifetime	lt		R	Lifetime of the registration in seconds
Sector	d	URI	RLA	Sector to which this endpoint belongs
Registration Base URI	base		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	L	Used for pagination
			RLA	Semantic name of the endpoint (see Section 9.4)

Table 2: RD Parameters

(Short: Short name used in query parameters or target attributes.  
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 has assigned the following multicast addresses for use by CoAP nodes:

IPv4 - "all CoRE resource directories" address, from the "IPv4 Multicast Address Space Registry" equal to "All CoAP Nodes", 224.0.1.187. 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.

#### 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.

#### 10.1.1.1. 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

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

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"
```

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
```

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 and Jaime Jimenez 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 -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-07 (work in progress), February 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-22 (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.silverajan-core-coap-protocol-negotiation]  
Silverajan, B. and M. Ocaik, "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
```

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"
```

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]"
```

## 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
```

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="/sensors/temp";  
  rel="describedby"
```

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
```

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]"
```

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"
```

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"
```

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](http://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: September 13, 2019

A. Keranen  
Ericsson  
M. Mohajer  
u-blox UK  
March 12, 2019

FETCH & PATCH with Sensor Measurement Lists (SenML)  
draft-ietf-core-senml-etch-03

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 September 13, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must



include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	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. Security Considerations . . . . .	5
5. IANA Considerations . . . . .	6
5.1. CoAP Content-Format Registration . . . . .	6
5.2. senml-etch+json Media Type . . . . .	6
5.3. senml-etch+cbor Media Type . . . . .	7
6. Acknowledgements . . . . .	8
7. References . . . . .	8
7.1. Normative References . . . . .	8
7.2. Informative References . . . . .	9
Authors' Addresses . . . . .	9

## 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::2/3306/0/", "n":"5850", "vb":true},
  {"n":"5851", "v":42},
  {"n":"5750", "vs":"Ceiling light"}
]
```

Here three resources "3306/0/5850", "3306/0/5851", and "3306/0/5750", 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::2/3306/0/", hence full names for resources are "2001:db8::2/3306/0/5850", 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::2/3306/0/", "n":"5850"},
  {"n":"5851"}
]
```

The result to a FETCH request with the example above would be:

```
[
  {"bn":"2001:db8::2/3306/0/", "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 field with value null, the matched Record (if any) is removed from the Pack.

For example, the following document could be given as (i)PATCH payload to change/set values of two SenML Records for the example in Section 1:

```
[
  {"bn":"2001:db8::2/3306/0/", "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::2/3306/0/", "n":"5850", "vb":false},
  {"n":"5851", "v":10},
  {"n":"5750", "vs":"Ceiling light"}
]
```

#### 4. 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.

## 5. 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.

### 5.1. 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]. All IDs are assigned from the "IETF Review or IESG Approval" range. The assigned IDs are show in Table 1.

Media type	ID
application/senml-etch+json	TBD
application/senml-etch+cbor	TBD

Table 1: CoAP Content-Format IDs

### 5.2. senml-etch+json Media Type

Type name: application

Subtype name: senml-etch+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. This simplifies implementation of a very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 4 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: N/A

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

### 5.3. senml-etch+cbor Media Type

Type name: application

Subtype name: senml-etch+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049].

Security considerations: See Section 4 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: N/A

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

## 6. 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, and other participants from the IETF CoRE and OMA SpecWorks DMSE working groups who have contributed ideas and reviews.

## 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>>.
- [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>>.

## 7.2. Informative References

- [IPSO] IPSO, "IPSO Smart Object Guidelines", 2018, <<https://www.omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/guidelines/>>.

## Authors' Addresses

Ari Keranen  
Ericsson

Email: [ari.keranen@ericsson.com](mailto:ari.keranen@ericsson.com)



Mojan Mohajer  
u-blox UK

Email: [Mojan.Mohajer@u-blox.com](mailto:Mojan.Mohajer@u-blox.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: June 22, 2019

M. Veillette, Ed.  
Trilliant Networks Inc.  
A. Pelov, Ed.  
I. Petrov, Ed.  
Acklio  
December 19, 2018

YANG Schema Item iDentifier (SID)  
draft-ietf-core-sid-05

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned numbers 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 June 22, 2019.

Copyright Notice

Copyright (c) 2018 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 . . . . .	7
5. Third party registries . . . . .	11
6. Security Considerations . . . . .	11
7. IANA Considerations . . . . .	11
7.1. Module registration . . . . .	12
7.2. "SID mega-range" registry . . . . .	12
7.2.1. "IANA SID Mega-Range" allocation . . . . .	13
7.2.2. "RFC SID range assignment" sub-registry . . . . .	14
7.2.3. "Specification SID range assignment" sub-registry . . . . .	14
7.3. "YANG module assignment" registry . . . . .	15
8. Acknowledgments . . . . .	15
9. References . . . . .	16
9.1. Normative References . . . . .	16
9.2. Informative References . . . . .	16
Appendix A. ".sid" file example . . . . .	17
Authors' Addresses . . . . .	26

## 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 part 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

To minimize their size, SIDs are often represented as a difference between the current SID and a reference SID. Such difference is called "delta", shorthand for "delta-encoded SID". Conversion from SIDs to deltas and back to SIDs is a stateless process. Each protocol implementing deltas must unambiguously define the reference SID for each YANG item.

SIDs are globally unique numbers, 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, 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 encoding is implemented.
4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. This process can also be automated using the same method described above, only unassigned YANG items are processed at step #3.

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 delta : Difference between the current SID and a reference SID. Each protocol that uses delta encoded SIDs MUST define how the reference SID is obtained.
- 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

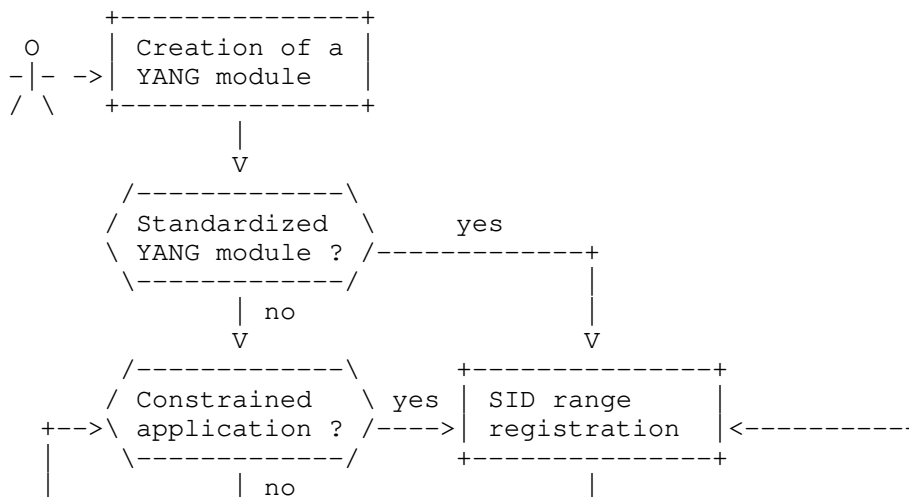
YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoMI [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

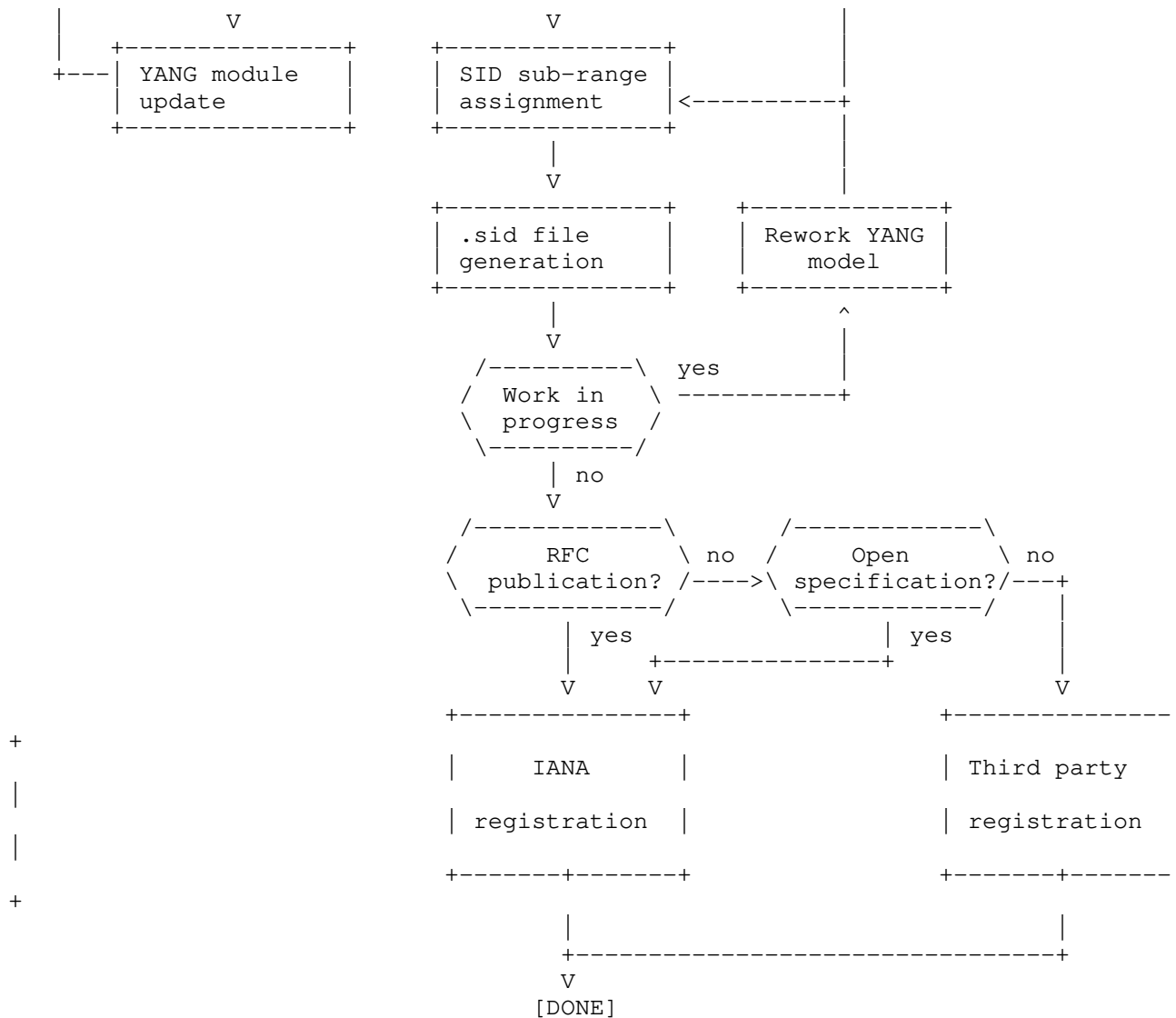
YANG modules are not necessarily created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

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. It could be "RFC SID range assignment" sub-registry as defined in Section 7.2.2, the "Specification SID range assignment" sub-registry as defined in Section 7.2.3 or another one, depending on the particular case. The minimal information required for this would be a start SID number and a range size, but might include additional details depending on the registry policy, which is outside the scope of this document. Once a SID range is registered, 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.

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 requirement for the registration and publication of the ".sid" files.

The following activity diagram summarizes the creation of a YANG module and its associated .sid file.

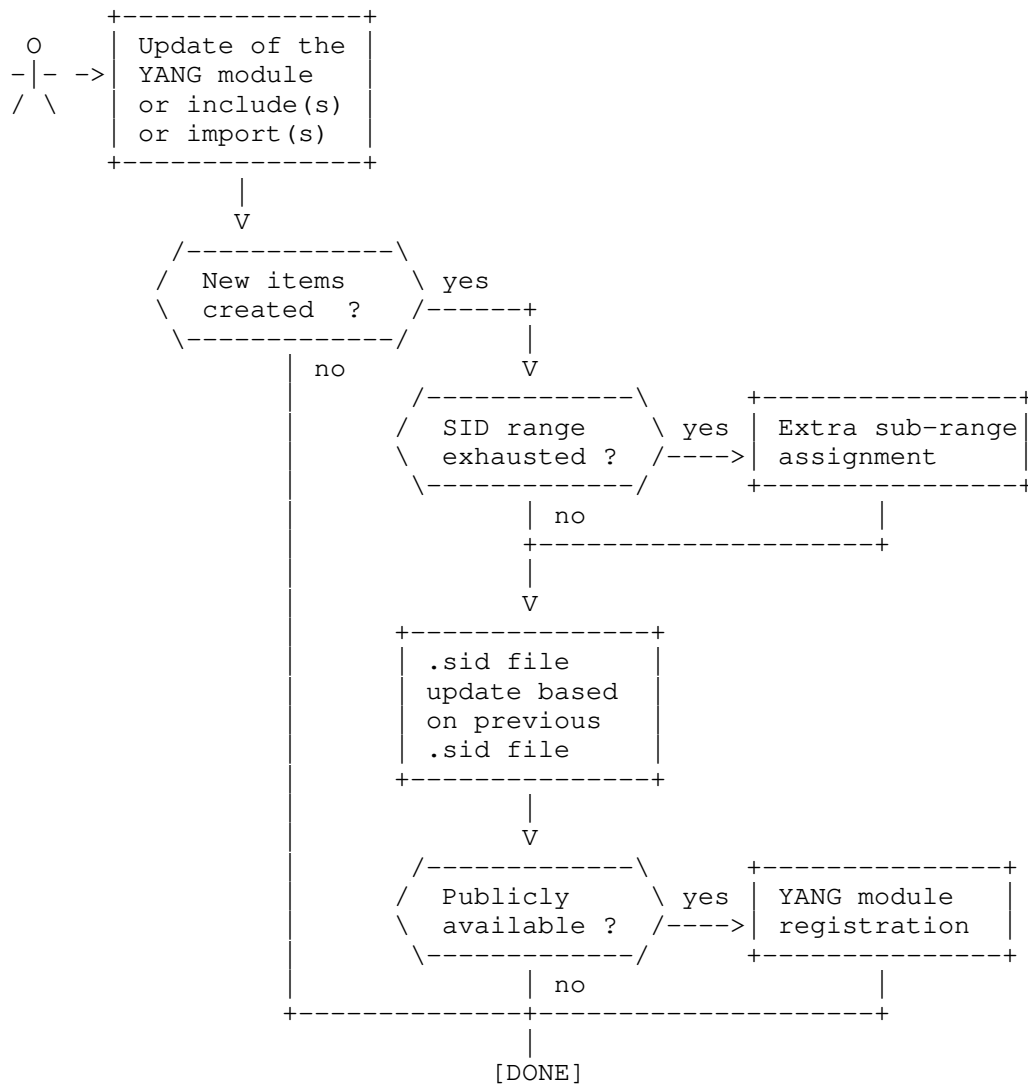




Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

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.

The following activity diagram summarizes the update of a YANG module and its associated .sid file.



#### 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;
}

import ietf-comi {
  prefix comi;
}

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 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 com: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 comi:sid;
  mandatory true;
  description
    "SID assigned to the YANG item for this mapping entry."
}
}
}
<CODE ENDS>
```

## 5. Third party registries

The organization and functioning of third party registries is outside the scope of the current document. The only limitations connected to those registries are listed in Section 7.2.

## 6. Security Considerations

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

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.

## 7. IANA Considerations

In this section are given specifications for an entry into the module registry and two new registries, a SID-range registry and a SID module registry.

### 7.1. Module registration

This document registers one YANG modules 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]]

### 7.2. "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 (e.g. SDO, registrar).

Each entry in this registry must include:

- o The entry point (first entry) of the registered SID range.
- o The size of the registered SID range.
- o The contact information of the requesting organization including:
  - o Organization name
  - o Primary contact name, email address, and phone number
  - o Secondary contact name, email address, and phone number

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Organization name
0	1000000	IANA

The IANA policies for future additions to this registry are "Hierarchical Allocation, Expert Review" [RFC5226]. Prior to a first allocation, the requesting organization must demonstrate a functional registry infrastructure. 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).

## 7.2.1. "IANA SID Mega-Range" allocation

The first million SIDs assigned to IANA is sub-divided as follow:

- o The range of 0 to 999 is reserved for future extensions. The IANA policy for this range is "IETF review" [RFC5226]. This range is reserved for a future uses and no sub-registries are currently defined for it.
- o The range of 1000 to 59,999 is reserved for YANG modules defined in RFCs. The IANA policy for future additions to this sub-registry is "RFC required" [RFC5226]. Allocation within this range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry. The allocation within this range is done during IESG review.
- o The range of 60,000 to 99,999 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" [RFC5226].
- o The range of 100,000 to 999,999 is reserved for standardized YANG modules. The IANA policy for future additions to this sub-registry is "Specification Required" [RFC5226]. Allocation within this range requires publishing of the associated ".yang" and ".sid" files in the YANG module registry.

Entry Point	Size	IANA policy
0	1,000	IETF review
1,000	59,000	RFC required
60,000	40,000	Experimental use
100,000	900,000	Specification Required

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 extra SID range can be allocated to an existing YANG module if the initial range is exhausted.

### 7.2.2. "RFC SID range assignment" sub-registry

The name of this sub-registry is "RFC SID range assignment". This sub-registry of "IANA SID Mega-Range" allocation Section 7.2.1 corresponds to the SID entry point 1000, size 59000. Each entry in this sub-registry must include:

- o The SID range entry point.
- o The SID range size.
- o The YANG module name.
- o The RFC number.

Initial entries in this registry are as follows:

Entry Point	Size	Module name	RFC number
1000	100	ietf-comi	[I-D.ietf-core-comi]
1100	50	ietf-yang-types	[RFC6021]
1150	50	ietf-inet-types	[RFC6021]
1200	50	iana-crypt-hash	[RFC7317]
1250	50	ietf-netconf-acm	[RFC6536]
1300	50	ietf-sid-file	RFCXXXX
1500	100	ietf-interfaces	[RFC7223]
1600	100	ietf-ip	[RFC7277]
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 module is required as per [RFC5226]. The YANG module must be registered in the "YANG module Name" registry according to the rules specified in section 14 of [RFC6020].

### 7.2.3. "Specification SID range assignment" sub-registry

The name of this sub-registry is "Specification SID range assignment". This sub-registry of "IANA SID Mega-Range" allocation Section 7.2.1 corresponds to the SID entry point 100000, size 900000. Each entry in this sub-registry must include:

- o The SID range entry point.

- o The SID range size.
- o The YANG module name.
- o The name of the standard organization
- o The specification identifier or URI

### 7.3. "YANG module assignment" registry

The name of this registry is "YANG module assignment". This registry is used to track which YANG modules have been assigned and the specific YANG items assignment. Each entry in this registry must include:

- o The YANG module name.
- o The associated ".yang" file(s)
- o The associated ".sid" file

The validity of the ".yang" and ".sid" files added to this registry MUST be verified.

- o The syntax of the registered ".yang" and ".sid" files must be valid.
- o Each YANG item defined by the registered ".yang" file must have a corresponding SID assigned in the ".sid" file.
- o Each SID is assigned to a single YANG item, duplicate assignment is not allowed.
- o The SID range(s) defined in the ".sid" file must be unique, must not conflict with any other SID ranges defined in already registered ".sid" files.
- o The ownership of the SID range(s) should be verified.

The IANA policy for future additions to this registry is "First Come First Served" as described in [RFC5226].

## 8. 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.



## 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>>.
- [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>>.
- [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>>.

### 9.2. Informative References

- [I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-04 (work in progress), November 2018.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <<https://www.rfc-editor.org/info/rfc5226>>.
- [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>>.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <<https://www.rfc-editor.org/info/rfc6021>>.
- [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>>.

- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [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>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<https://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [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>>.

#### 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
}
]
```

#### Authors' Addresses

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

Phone: +14503750556  
Email: [michel.veillette@trilliant.com](mailto:michel.veillette@trilliant.com)

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

Email: [a@ackl.io](mailto:a@ackl.io)

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

Email: [ivaylo@ackl.io](mailto:ivaylo@ackl.io)

CoRE Working Group  
Internet-Draft  
Updates: 7252, 8323 (if approved)  
Intended status: Standards Track  
Expires: September 12, 2019

K. Hartke  
Ericsson  
March 11, 2019

Extended Tokens and Stateless Clients  
in the Constrained Application Protocol (CoAP)  
draft-ietf-core-stateless-01

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 September 12, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	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-Lengths Capability Option . . . . .	5
2.2.2. Trial and Error . . . . .	5
2.3. Intermediaries . . . . .	6
3. Stateless Clients . . . . .	7
3.1. Intermediaries . . . . .	7
3.2. Extended Tokens . . . . .	8
3.3. Message Transmission . . . . .	9
4. Security Considerations . . . . .	10
4.1. Extended Tokens . . . . .	10
4.2. Stateless Clients . . . . .	10
4.2.1. Recommended Algorithms . . . . .	11
5. IANA Considerations . . . . .	11
5.1. CoAP Signaling Option Number . . . . .	11
6. References . . . . .	11
6.1. Normative References . . . . .	11
6.2. Informative References . . . . .	12
Appendix A. Updated Message Formats . . . . .	12
A.1. CoAP over UDP . . . . .	13
A.2. CoAP over TCP . . . . .	14
A.3. CoAP over WebSockets . . . . .	15
Acknowledgements . . . . .	16
Author's Address . . . . .	16

## 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 serve 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. The server returns the token verbatim in any resulting CoAP response (Figure 1).

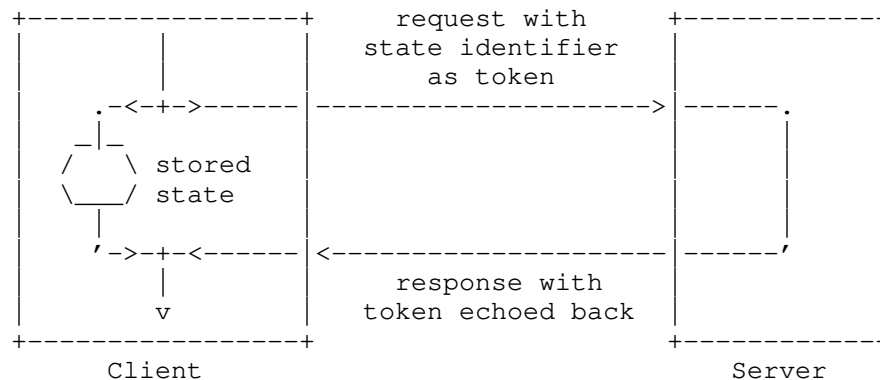


Figure 1: Token as an Identifier for Request State

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. Clients can implement this by serializing (parts of) their state into the token itself and recovering the state from the token in the response (Figure 2).

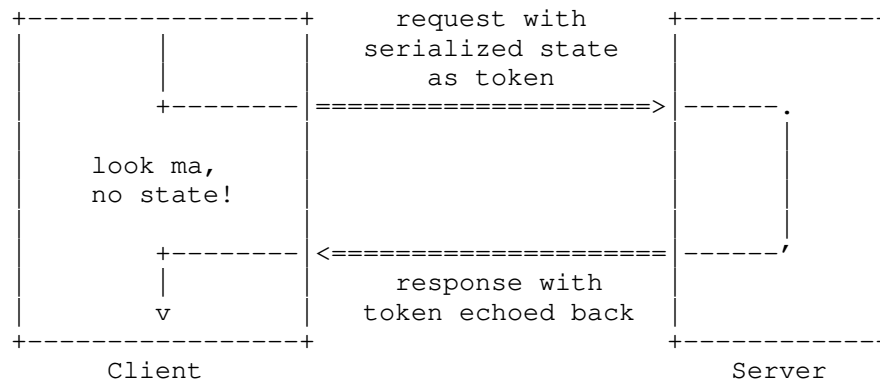


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for making clients "stateless" in this way, i.e., for avoiding per-request state in client implementations. (They'll still need to maintain per-server state and other kinds of state, so they're not entirely stateless.)

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

### Stateless

In this document, "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that doesn't 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 (such as state for generating tokens and congestion control), so it's not free of any state.

### Client

In this document, "client" generally refers to any sender of a request and recipient of a response, including intermediaries.

### Server

In this document, "server" generally refers to any recipient of a request and sender of a response, including intermediaries.

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

### 2.1. Extended Token Length (TKL) Field

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the following new definition of the TKL field, increasing the maximum token length to 65804 bytes.

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.

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 or, if there are one or more intermediaries between the client and the server, the intermediary in the server role that the client is interacting with.

Support can be discovered by a client (or intermediary in the client role) in one of two ways: In case Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Lengths Capability Option defined in Section 2.2.1. 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-Lengths Capability Option

A sender can use the elective Extended-Token-Lengths Capability Option to indicate its support for the new TKL field definition specified in Section 2.1.

#	C	R	Appli es to	Name	Forma t	Length	Base Value
TB D			CSM	Extended-Token- Lengths	empty	0	(none)

C=Critical, R=Repeatable

Table 1: The Extended-Token-Lengths Capability Option

### 2.2.2. Trial and Error

A request with a TKL field value outside the range from 0 to 8 will be considered a message format error (Section 3 of RFC 7252) and be rejected by a recipient that does not support the updated TKL field definition. A client thus can determine support by sending a request



with an extended token length and checking whether it's rejected by the recipient or not.

In CoAP over UDP, a recipient rejects a malformed confirmable message by sending a Reset message (Section 4.2 of RFC 7252). In case of a non-confirmable message, sending a Reset message is permitted but not required (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 don't contain a token; they only return the Message ID (Figure 3). They also don't contain any indication of what caused a message format error. It is therefore RECOMMENDED that clients use a request that contains no potential message format error other than the extended token length.

In CoAP over TCP, TLS, and WebSockets, a recipient rejects a malformed message by sending an Abort message and shutting down the connection (Section 5.6 of RFC 8323).

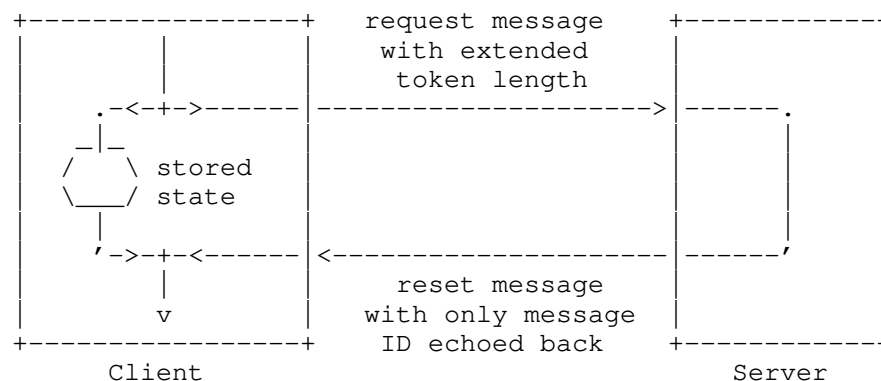


Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Next Hop Does Not Support It

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to process, it MUST NOT reject the message. Instead, it SHOULD return a 4.00 (Bad Request) response. This implies that the server returns the entire token verbatim.

### 2.3. Intermediaries

Tokens are a hop-by-hop feature: 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 when receiving a request with an extended token length.

### 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 will return the token in the response to the client, so that the client can recover the state and process the response as if it had kept the state locally.

The format of the serialized state is an implementation detail of the client and opaque to any server implementation. However, using tokens to serialize state has significant and non-obvious security and privacy implications that need to be mitigated; see Section 4.

#### 3.1. 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 to the next hop 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 ongoing 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 ongoing requests. This can be implemented by serializing this information along the request state into the token to the next hop. When the next hop returns the response, the intermediary can recover the information from the token and use it to satisfy the client's request.

The downside of this approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which reduces efficiency.

When multiple clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of RFC 7641). As this cannot be satisfied without keeping request state, an intermediary MUST NOT include an Observe Option in requests it sends without keeping request state.

When using blockwise 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 blockwise transfers with a value that uniquely identifies the next hop towards the client in the intermediary's namespace.

### 3.2. Extended Tokens

A client (or intermediary in the role of a client) that depends on support for extended token lengths (Section 2) from the next hop to avoid keeping request state **MUST** 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 next hop doesn't support extended tokens, then any error message couldn't be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

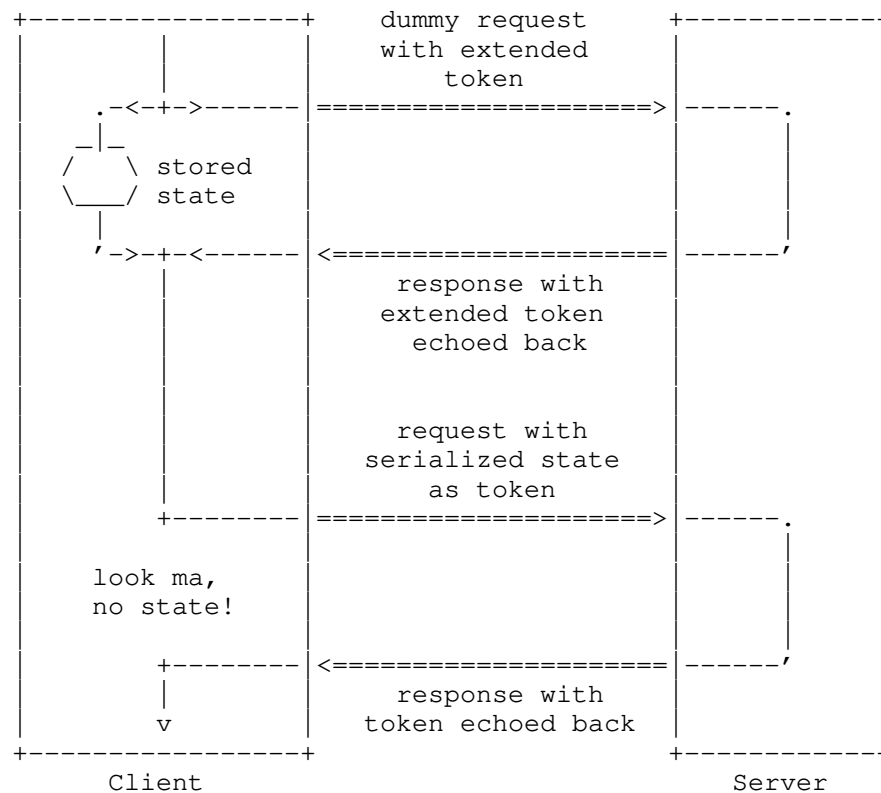


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

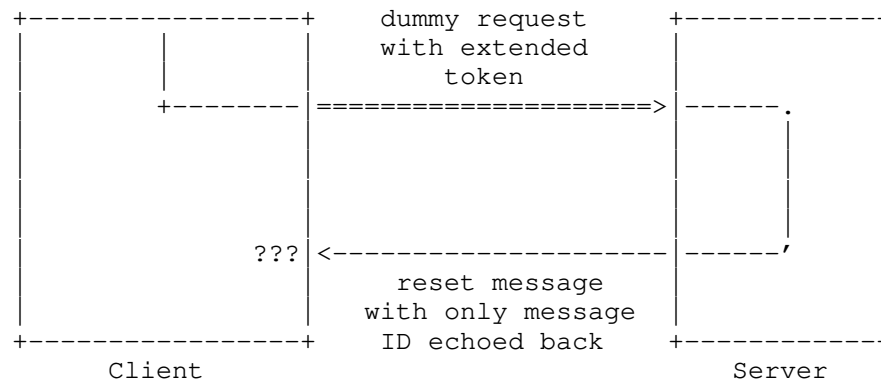


Figure 5: Stateless Discovery of Support Does Not Work

### 3.3. Message Transmission

As a further step, in the case of CoAP over UDP [RFC7252], a client (or intermediary in the client role) might want to also avoid keeping message transmission state.

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 needs to handle these response types as follows:

- o If a piggybacked response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST silently ignore the message.
- o If a separate response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject the message as specified in Section 4.2 of RFC 7252.

- o If a non-confirmable response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject 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 for nodes in particular 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 MUST be prepared for this and mitigate it.

### 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.

Serialized state information is an attractive target for both unwanted nodes (attackers between the node in client role and the next hop) and wanted nodes (the next hop itself) on the path. Therefore, a node in the client role MUST integrity protect the state information, unless processing a response does not modify state or cause 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 MUST implement replay protection (e.g., by using sequence numbers and a replay window), unless processing a response does not modify state or cause other 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 MUST include freshness information (e.g., a timestamp).

Information in the serialized state may be privacy sensitive. A node in client role MUST encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise. For example, an intermediary that serializes the client's

token and transport address into its token leaks that information to the next hop, which may be undesirable. In wireless mesh networks, where all traffic is visible to a passive attacker, encryption may not be needed as the attacker can get the same information from analyzing the traffic flows.

#### 4.2.1. Recommended Algorithms

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.

### 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-Lengths	[[this document]]

### 6. References

#### 6.1. Normative References

- [I-D.ietf-core-echo-request-tag]  
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [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>>.

- [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>>.

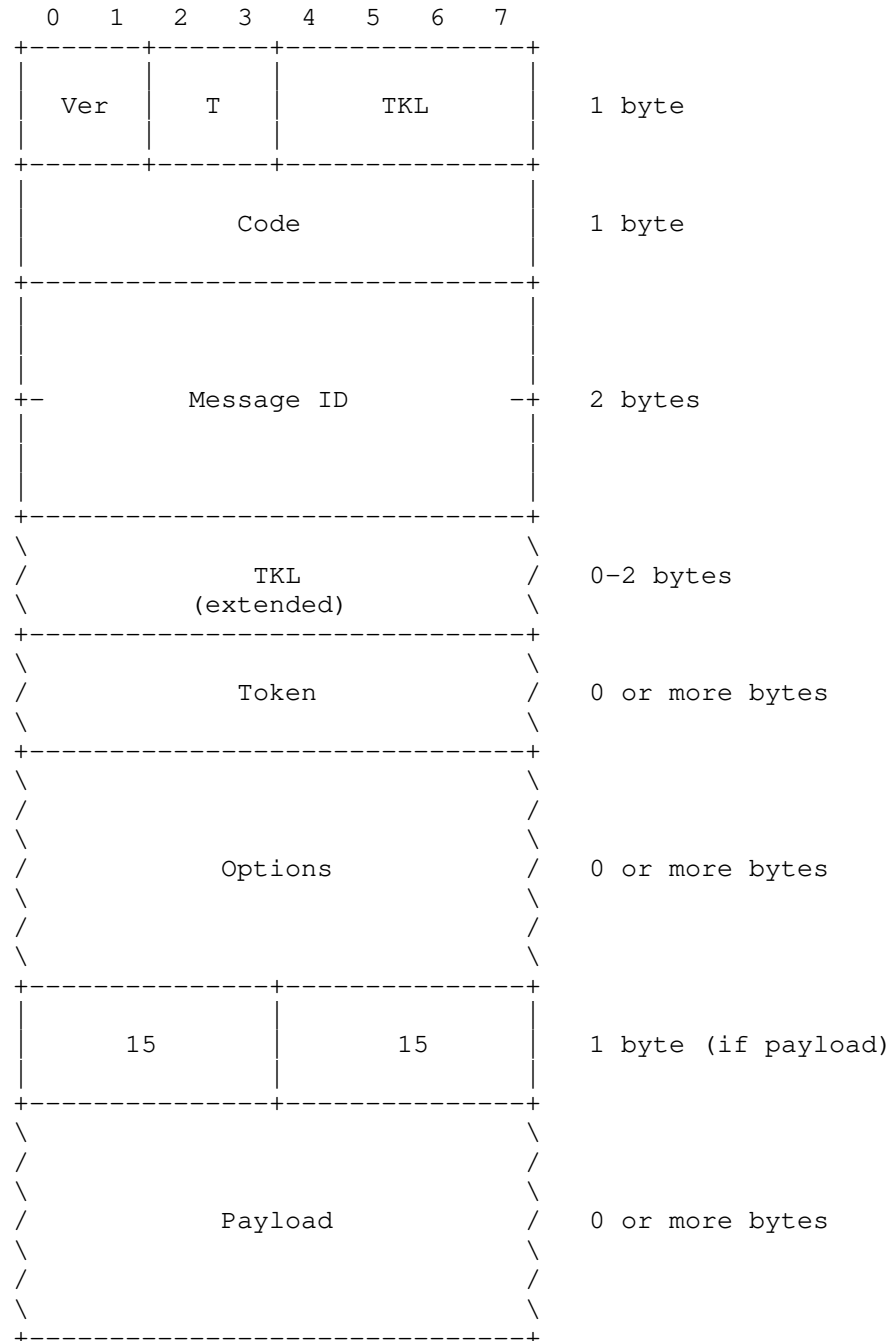
## 6.2. Informative References

- [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-09 (work in progress), November 2018.
- [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

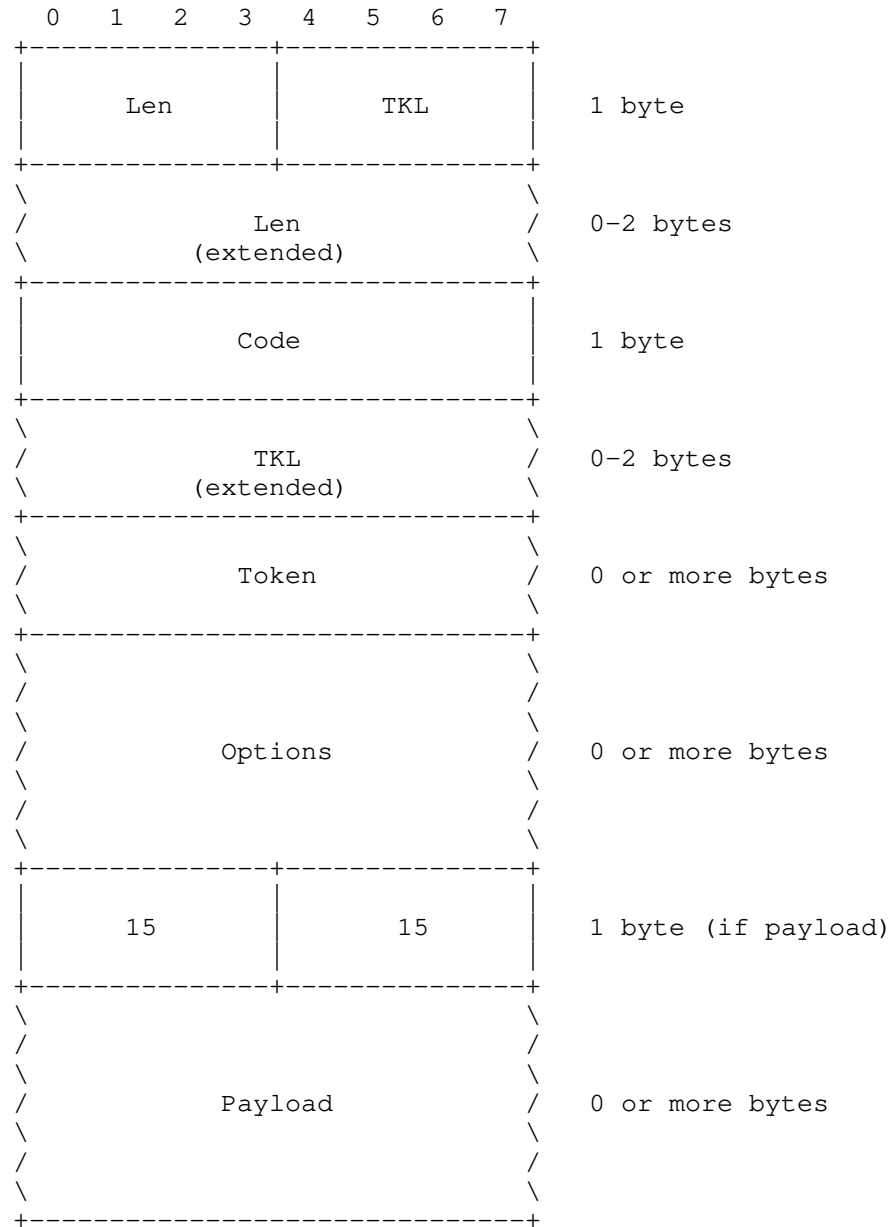
This appendix illustrates the CoAP message formats updated with the new definition of the TKL field (Section 2).

# A.1. CoAP over UDP

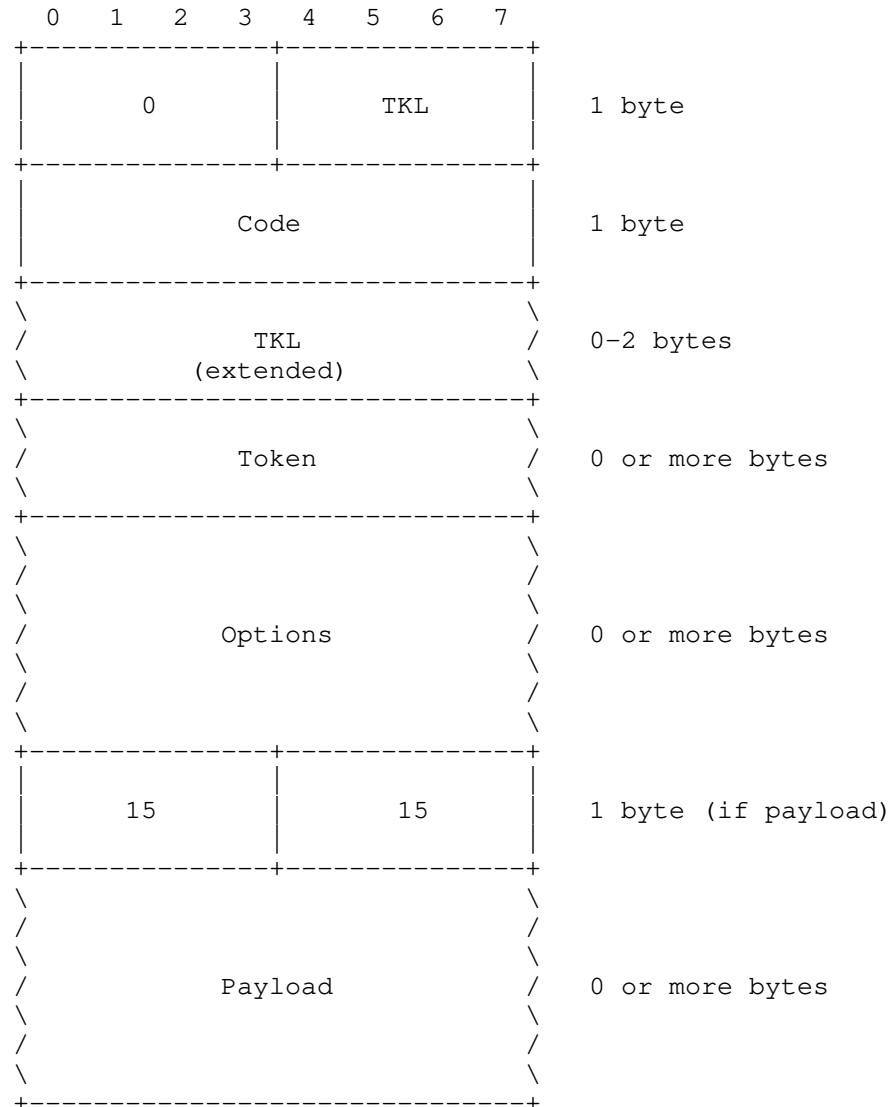




## A.2. CoAP over TCP



### 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 Carsten Bormann, Ari Keranen, John Mattsson, Jim Schaad, Goeran Selander, and Malisa Vucinic 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

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: March 18, 2019

M. Veillette, Ed.  
Trilliant Networks Inc.  
A. Pelov, Ed.  
Acklio  
A. Somaraju  
Tridonic GmbH & Co KG  
R. Turner  
Landis+Gyr  
A. Minaburo  
Acklio  
September 14, 2018

CBOR Encoding of Data Modeled with YANG  
draft-ietf-core-yang-cbor-07

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 18, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

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

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

## 1. Introduction

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

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

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

## 2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- 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 [RFC7951]:

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

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.

## 2.1. YANG Schema Item iDentifier (SID)

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

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

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

## 2.2. CBOR diagnostic notation

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



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

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- o Any text within and including a pair of slashes is considered a comment.
- o Deltas are visualized as numbers preceded by a '+' or '-' sign. The use of the '+' sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

### 3. Properties of the CBOR Encoding

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

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

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

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

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

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

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

#### 4. Encoding of YANG Schema Node Instances

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

##### 4.1. The 'leaf'

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

#### 4.2. The 'container' and other collections

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

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

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

Definition example from [RFC7317]:

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

container system-state {

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

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

##### 4.2.1. SIDs as keys

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

- 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 an 'rpc input' or 'rcp 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'.

CBOR diagnostic notation:

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

#### 4.2.2. Member names as keys

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

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

Definition example from [RFC7317]:

```

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

```

```

container system-state {

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

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

```

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

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

```

#### 4.3. The 'leaf-list'

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

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

Definition example [RFC7317]:

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

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

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

CBOR encoding: 82 68 696574662E6F7267 68 696565652E6F7267

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

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

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

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

Definition example from [RFC7317]:

```

list server {
    key name;

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

```

#### 4.4.1. SIDs as keys

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

CBOR diagnostic notation:

```

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

```

CBOR encoding:

```

82          # array(2)
  A5        # map(5)
    03      # unsigned(3)
    6E      # text(14)
      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. Member names as keys

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

CBOR diagnostic notation:

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

CBOR encoding:

```

82                                     # array(2)
  A5                                 # map(5)
    70                             # text(16)
      696574662D73797374656D3A6E616D65 # "ietf-system:name"
    6E                             # text(14)
      4E52432054494320736572766572    # "NRC TIC server"
    6F                             # text(15)
      696574662D73797374656D3A756470 # "ietf-system:udp"
  A2                                 # map(2)
    67                             # text(7)
      61646472657373                  # "address"
    6A                             # text(10)
      74696332E6E72632E6361          # "tic.nrc.ca"
    64                             # text(4)
      706F7274                        # "port"
    18 7B                          # unsigned(123)
  78 1C                             # text(28)
      696574662D73797374656D3A6173736F636961746966F6E2D74797065
  00                               # unsigned(0)
  72                               # text(18)
      696574662D73797374656D3A696275727374 # "ietf-system:iburst"
  F4                               # primitive(20)
  72                               # text(18)
      696574662D73797374656D3A707265666572 # "ietf-system:prefer"
  F5                               # primitive(21)
  A2                                 # map(2)
    70                             # text(16)
      696574662D73797374656D3A6E616D65 # "ietf-system:name"
    6E                             # text(14)
      4E52432054414320736572766572    # "NRC TAC server"
    6F                             # text(15)
      696574662D73797374656D3A756470 # "ietf-system:udp"
  A1                                 # map(1)
    67                             # text(7)
      61646472657373                  # "address"
    6A                             # text(10)
      74616332E6E72632E6361          # "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 member 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 event;                               # SID 60123
```

This example also assumes the assistance of the following notification.

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

CBOR diagnostic notation:

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

CBOR encoding:

```

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

```

#### 4.6. The 'anyxml'

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

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

Definition example from [RFC7951]:

```
anyxml bar;
```

CBOR diagnostic notation: [true, null, true]

CBOR encoding: 83 f5 f6 f5

#### 5. Encoding of YANG data templates

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

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

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

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

```

import ietf-restconf {
  prefix rc;
}

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

```

### 5.1. SIDs as keys

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

CBOR diagnostic notation:

```

{
  1024 : {
    +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)
    4D6178696D756D206578636565646564

```

## 5.2. Member names as keys

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

CBOR diagnostic notation:

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

CBOR encoding:

```

A1                                # map(1)
  6F                              # text(15)
    696574662D636F6D693A6572726F72  # "ietf-comi:error"
A4                                # map(4)
  69                              # text(9)
    6572726F722D746167              # "error-tag"
  6D                              # text(13)
    696E76616C69642D76616C7565      # "invalid-value"
  6D                              # text(13)
    6572726F722D6170702D746167      # "error-app-tag"
  6C                              # text(12)
    6E6F742D696E2D72616E6765        # "not-in-range"
  6F                              # text(15)
    6572726F722D646174612D6E6F6465  # "error-data-node"
  73                              # text(19)
    74696D657A6F6E652D7574632D6F6666736574 # "timezone-utc-offset"
  6D                              # text(13)
    6572726F722D6D657373616765      # "error-message"
  70                              # text(16)
    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 [RFC7277]:

```

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

```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

## 6.2. The integer Types

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: -300

CBOR encoding: 39 012B

## 6.3. The 'decimal64' Type

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

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

Definition example from [RFC7317]:

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

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

CBOR encoding: C4 82 21 19 0101



#### 6.4. The 'string' Type

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

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

Definition example from [RFC7223]:

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

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

#### 6.5. The 'boolean' Type

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: true

CBOR encoding: F5

#### 6.6. The 'enumeration' Type

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

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

Definition example from [RFC7317]:

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

CBOR diagnostic notation: 3

CBOR encoding: 03

## 6.7. The 'bits' Type

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

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

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

Definition example from [RFC7950]:

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

CBOR diagnostic notation: h'05'

CBOR encoding: 41 05

#### 6.8. The 'binary' Type

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

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

Definition example:

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

CBOR diagnostic notation: h'1F1CE6A3F42660D888D92A4D8030476E'

CBOR encoding: 50 1F1CE6A3F42660D888D92A4D8030476E

#### 6.9. The 'leafref' Type

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

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

Definition example from [RFC7223]:

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

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

#### 6.10. The 'identityref' Type

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

##### 6.10.1. SIDs as identityref

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

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

Definition example from [RFC7317]:

```

identity interface-type {
}

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

identity ethernetCsmacd {
    base iana-interface-type;
}

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

```

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

#### 6.10.2. Name as identityref

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

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

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

CBOR encoding: 78 1b

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 [RFC7277]:

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

CBOR diagnostic notation: null

CBOR encoding: F6

#### 6.12. The 'union' Type

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

- o bits
- o enumeration
- o identityref
- o instance-identifier

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

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

Definition example from [RFC7317]:

```

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

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

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

leaf address {
  type inet:ip-address;
}

```

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

CBOR encoding: 74 323030313A6462383A6130623A313266303A3A31

### 6.13. The 'instance-identifier' Type

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

#### 6.13.1. SIDs as instance-identifier

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

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

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

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

- 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

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

\*First example:\*

This example is described in Section 6.13.1.

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

CBOR encoding:

```
78 1c 2F6965744662D73797374656D3A73797374656D2F636F6E74616374
```

\*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
  2F6965746662D73797374656D3A73797374656D2F61757468656E74696361
  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
  2F6965746662D73797374656D3A73797374656D2F61757468656E74696361
  74696F6E2F757365725B6E616D653D27626F62275D
```

## 7. Security Considerations

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

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

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

## 8. IANA Considerations

### 8.1. Tags Registry

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

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

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

## 9. Acknowledgments

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

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

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.

## 10.2. Informative References

- [I-D.ietf-core-comi]  
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-03 (work in progress), June 2018.
- [I-D.ietf-core-sid]  
Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-04 (work in progress), June 2018.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<https://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

Authors' Addresses

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

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

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

Email: a@ackl.io

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

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

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

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

Ana Minaburo  
Acklio  
2bis rue de la chataigneraie  
Cesson-Sevigne, Bretagne 35510  
France

Email: ana@ackl.io

CoRE Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: April 25, 2019

I. Jarvinen  
M. Kojo  
I. Raitahila  
University of Helsinki  
Z. Cao  
Huawei  
October 22, 2018

Fast-Slow Retransmission Timeout and Congestion Control Algorithm for  
CoAP  
draft-jarvinen-core-fasor-01

Abstract

This document specifies an alternative retransmission timeout and congestion control back off algorithm for the CoAP protocol, called Fast-Slow RTO (FASOR).

The algorithm specified in this document employs an appropriate and large enough back off of Retransmission Timeout (RTO) as the major congestion control mechanism to allow acquiring unambiguous RTT samples with high probability and to prevent building a persistent queue when retransmitting. The algorithm also aims to retransmit quickly using an accurately managed retransmission timeout when link-errors are occurring, basing RTO calculation on unambiguous round-trip time (RTT) samples.

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 25, 2019.

## Copyright Notice

Copyright (c) 2018 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. Conventions . . . . .	3
3. Problems with Existing CoAP Congestion Control Algorithms . .	3
4. FASOR Algorithm . . . . .	4
4.1. Computing Normal RTO (FastRTO) . . . . .	4
4.2. Slow RTO . . . . .	5
4.3. Retransmission Timeout Back Off Logic . . . . .	6
4.3.1. Overview . . . . .	6
4.3.2. Retransmission State Machine . . . . .	7
4.4. Retransmission Count Option . . . . .	9
4.5. Alternatives for Exchanging Retransmission Count Information . . . . .	11
5. Security Considerations . . . . .	11
6. IANA Considerations . . . . .	11
7. References . . . . .	11
7.1. Normative References . . . . .	11
7.2. Informative References . . . . .	12
Appendix A. Pseudocode for Basic FASOR without Dithering . . . .	13
Authors' Addresses . . . . .	15

## 1. Introduction

CoAP senders use retransmission timeout (RTO) to infer losses that have occurred in the network. For such a heuristic to be correct, the RTT estimate used for calculating the retransmission timeout must match to the real end-to-end path characteristics. Otherwise, unnecessary retransmission may occur. Both default RTO mechanism for CoAP [RFC7252] and CoCoA [I-D.ietf-core-cocoa] have issues in dealing with unnecessary retransmissions and in the worst-case the situation can persist causing congestion collapse [JRCK18a].



This document specifies FASOR retransmission timeout and congestion control algorithm [JRCK18b]. FASOR algorithm ensures unnecessary retransmissions that a sender may have sent due to an inaccurate RTT estimate will not persist avoiding the threat of congestion collapse. FASOR also aims to quickly restore the accuracy of the RTT estimate. Armed with an accurate RTT estimate, FASOR not only handles congestion robustly but also can quickly infer losses due to link errors.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

## 3. Problems with Existing CoAP Congestion Control Algorithms

Correctly inferring losses requires the retransmission timeout (RTO) to be longer than the real RTT in the network. Under certain circumstances the RTO may be incorrectly small. If the real end-to-end RTT is larger than the retransmission timeout, it is impossible for the sender to avoid making unnecessary retransmissions that duplicate data still existing in the network because the sender cannot receive any feedback in time. Unnecessary retransmissions cause two basic problems. First, they increase the perceived end-to-end RTT if the bottleneck has buffering capacity, and second, they prevent getting unambiguous RTT samples. Making unnecessary retransmissions is also a pre-condition for the congestion collapse [RFC0896], which may occur in the worst case if retransmissions are not well controlled [JRCK18a]. Therefore, the sender retransmission timeout algorithm should actively attempt to prevent unnecessary retransmissions from persisting under any circumstance.

Karn's algorithm [KP87] has prevented unnecessary retransmission from turning into congestion collapse for decades due to robust RTT estimation and retransmission timeout backoff handling. The recent CoAP congestion control algorithms, however, diverge from the principles of Karn's algorithm in significant ways and may pose a threat to the stability of the Internet due to those differences.

The default RTO mechanism for CoAP [RFC7252] uses only an initial RTO dithered between 2 and 3 seconds, while CoCoA [I-D.ietf-core-cocoa] measures RTT both from unambiguous and ambiguous RTT samples and applies a modified version of the TCP RTO algorithm [RFC6298]. The algorithm in RFC 7252 lacks solution to persistent congestion. The binary exponential back off used for the retransmission timeout does not properly address unnecessary retransmissions when RTT is larger

than the default RTO (ACK\_TIMEOUT). If the CoAP sender performs exchanges over an end-to-end path with such a high RTT, it persistently keeps making unnecessary retransmissions for every exchange wasting some fraction of the used resources (network capacity, battery power).

CoCoA [I-D.ietf-core-cocoa] attempts to improve scenarios with link-error related losses and solve persistent congestion by basing its RTO value on an estimated RTT. However, there are couple of exceptions when the RTT estimation is not available:

- At the beginning of a flow where initial RTO of 2 seconds is used.
- When RTT suddenly jumps high enough to trigger the rule in CoCoA that prevents taking RTT samples when more than two retransmissions are needed. This may also occur when the packet drop rate on the path is high enough.

When RTT estimate is too small, unnecessary retransmission will occur also with CoCoA. CoCoA being unable to take RTT samples at all is a particularly problematic phenomenon as it is similarly persisting state as with the algorithm outlined in RFC 7252 and the network remains in a congestion collapsed state due to persisting unnecessary retransmissions.

#### 4. FASOR Algorithm

FASOR [JRCK18b] is composed of three key components: RTO computation, Slow RTO, and novel retransmission timeout back off logic.

##### 4.1. Computing Normal RTO (FastRTO)

The FASOR algorithm measures the RTT for an CoAP message exchange over an end-to-end path and computes the RTO value using the TCP RTO algorithm specified in [RFC6298]. We call this normal RTO or FastRTO. In contrast to the TCP RTO mechanism, FASOR SHOULD NOT use 1 second lower-bound when setting the RTO because RTO is only a backup mechanisms for loss detection with TCP, whereas with CoAP RTO is the primary and only loss detection mechanism. A lower-bound of 1 second would impact timeliness of the loss detection in low RTT environments. The RTO value MAY be upper-bounded by at least 60 seconds. A CoAP sender using the FASOR algorithm SHOULD set initial RTO to 2 seconds. The computed RTO value as well as the initial RTO value is subject to dithering; they are dithered between  $RTO + 1/4 \times SRTT$  and  $RTO + SRTT$ . For dithering initial RTO, SRTT is unset; therefore, SRTT is replaced with initial RTO / 3 which is derived from the RTO formula and equals to a hypotheticalal initial RTT that

would yield the initial RTO using the SRTT and RTTVAR initialization rule of RFC 6298. That is, for initial RTO of 2 seconds we use SRTT value of  $2/3$  seconds.

FastRTO is updated only with unambiguous RTT samples. Therefore, it closely tracks the actual RTT of the network and can quickly trigger a retransmission when the network state is not dubious. Retransmitting without extra delay is very useful when the end-to-end path is subject to losses that are unrelated to congestion. When the first unambiguous RTT sample is received, the RTT estimator is initialized with that sample as specified in [RFC6298] except RTTVAR that is set to  $R/2K$ .

#### 4.2. Slow RTO

We introduce Slow RTO as a safe way to ensure that only a unique copy of message is sent before at least one RTT has elapsed. To achieve this the sender must ensure that its retransmission timeout is set to a value that is larger than the path end-to-end RTT that may be inflated by the unnecessary retransmission themselves. Therefore, whenever a message needs to be retransmitted, we measure Slow RTO as the elapsed time required for getting an acknowledgement. That is, Slow RTO is measured starting from the original transmission of the request message until the receipt of the acknowledgement, regardless of the number of retransmissions. In this way, Slow RTO always covers the worst-case RTT during which a number of unnecessary retransmissions were made but the acknowledgement is received for the original transmission. In contrast to computing normal RTO, Slow RTO is not smoothed because it is derived from the sending pattern of the retransmissions (that may turn out unnecessary). In order to drain the potential unnecessary retransmissions successfully from the network, it makes sense to wait for the time used for sending them rather than some smoothed value. However, Slow RTO is multiplied by a factor to allow some growth in load without making Slow RTO too aggressive (by default the factor of 1.5 is used). FASOR then applies Slow RTO as one of the backed off timer values used with the next request message.

Slow RTO allows rapidly converging towards stable operating point because 1) it lets the duplicate copies sent earlier to drain from the network reducing the perceived end-to-end RTT, and 2) allows enough time to acquire an unambiguous RTT sample for the RTO computation. Robustly acquiring the RTT sample ensures that the next RTO is set according to the recent measurement and further unnecessary retransmissions are avoided. Slow RTO itself is a form of back off because it includes the accumulated time from the retransmission timeout back off of the previous exchange. FASOR uses this for its advantage as the time included into Slow RTO is what is

needed to drain all unnecessary retransmissions possibly made during the previous exchange. Assuming a stable RTT and that all of the retransmissions were unnecessary, the time to drain them is the time elapsed from the original transmission to the sending time of the last retransmission plus one RTT. When the acknowledgement for the original transmission arrives, one RTT has already elapsed, leaving only the sending time difference still unaccounted for which is at minimum the value for Slow RTO (when an RTT sample arrives immediately after the last retransmission). Even if RTT would be increasing, the draining still occurs rapidly due to exponentially backed off frequency in sending the unnecessary retransmissions.

#### 4.3. Retransmission Timeout Back Off Logic

##### 4.3.1. Overview

FASOR uses normal RTO as the base for binary exponential back off when no retransmission were needed for the previous CoAP message exchange. When retransmission were needed for the previous CoAP message exchange, the algorithm rules, however, are more complicated than with the traditional RTO back off because Slow RTO is injected into the back off series to reduce high impact of using Slow RTO. FASOR logic chooses from three possible back off series alternatives:

**FAST back off:** Perform traditional RTO back off with the normal RTO as the base. Applied when the previous message was not retransmitted.

**FAST\_SLOW\_FAST back off:** First perform a probe using the normal RTO for the original transmission of the request message to improve cases with losses unrelated to congestion. If the probe for the original transmission of the request message is successful without retransmissions, continue with FAST back off for the next message exchange. If the request message needs to be retransmitted, continue by using Slow RTO for the first retransmission in order to respond to congestion and drain the network from the unnecessary retransmissions that were potentially sent for the previous exchange. If still further RTOs are needed, continue by backing off the normal RTO further on each timeout. FAST\_SLOW\_FAST back off is applied just once when the previous request message using FAST back off required one or more retransmissions.

**SLOW\_FAST back off:** Perform Slow RTO first for the original transmission to respond to congestion and to acquire an unambiguous RTT sample with high probability. Then, if the original request needs to be retransmitted, continue with the normal RTO-based RTO back off serie by backing off the normal RTO

on each timeout. SLOW\_FAST back off is applied when the previous request message using FAST\_SLOW\_FAST or SLOW\_FAST back off required one or more retransmissions. Once an acknowledgement for the original transmission with unambiguous RTT sample is received, continue with FAST back off for the next message exchange.

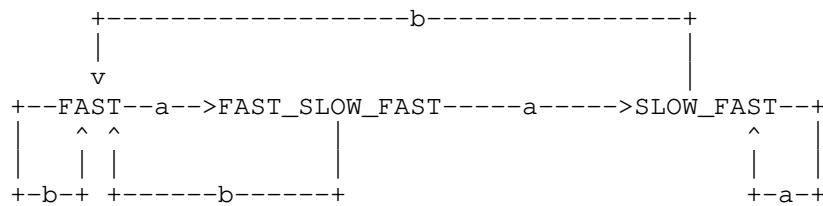
For the initial message, FAST is used with INITIAL\_RTO as the FastRTO value. From there on, state is updated when an acknowledgement arrives. Following unambiguous RTT samples, FASOR always uses FAST. Whenever retransmissions are needed, the back off series selection is first downgraded to FAST\_SLOW\_FAST back off and then to SLOW\_FAST back off if further retransmission are needed in FAST\_SLOW\_FAST.

When Slow RTO is used as the first RTO value, the sender is likely to acquire unambiguous RTT sample even when the network has high delay due to congestion because Slow RTO is based on a very recent measurement of the worst-case RTT. However, using Slow RTO may negatively impact the performance when losses unrelated to congestion are occurring. Due to its potential high cost, FASOR algorithm attempts to avoid using Slow RTO unnecessarily.

The CoAP protocol is often used by devices that are connected through a wireless network where non-congestion related losses are much more frequent than in their wired counterparts. This has implications for the retransmission timeout algorithm. While it would be possible to implement FASOR such that it immediately uses Slow RTO when a dubious network state is detected, which would handle congestion very well, it would do significant harm for performance when RTOs occur due to non-congestion related losses. Instead, FASOR uses first normal RTO for one transmission and only responds using Slow RTO if RTO expires also for that request message. Such a pattern quickly probes if the losses were unrelated to congestion and only slightly delays response if real congestion event is taking place. To ensure that an unambiguous RTT sample is also acquired on a congested network path, FASOR then needs to use Slow RTO for the original transmission of the subsequent packet if the probe was not successful.

#### 4.3.2. Retransmission State Machine

FASOR consists of the three states discussed above while making retransmission decisions, FAST, FAST\_SLOW\_FAST and SLOW\_FAST. The state machine of the FASOR algorithm is depicted in Figure 1.



a: retransmission acknowledged, ambiguous RTT sample acquired;  
b: no retransmission, unambiguous RTT sample acquired;

Figure 1: State Machine of FASOR

In the FAST state, if the original transmission of the message has not been acknowledged by the receiver within the time defined by `FastRTO`, the sender will retransmit it. If there is still no acknowledgement of the retransmitted packet within  $2 * \text{FastRTO}$ , the sender performs the second retransmission and if necessary, each further retransmission applying binary exponential back off of `FastRTO`. The retransmission interval in this state is defined as `FastRTO`,  $2^1 * \text{FastRTO}$ , ...,  $2^i * \text{FastRTO}$ .

When there is an acknowledgement after any retransmission, the sender will calculate `SlowRTO` value based on the algorithm defined in Section 4.2.

When there is an acknowledgement after any retransmission, the sender will also switch to the second state, `FAST_FLOW_FAST`. In this state, the retransmission interval is defined as `FastRTO`,  $\text{Max}(\text{SlowRTO}, 2 * \text{FastRTO})$ ,  $\text{FastRTO} * 2^1$ , ...,  $2^i * \text{FastRTO}$ . The state will be switched back to the FAST state once an acknowledgement is returned within `FastRTO`, i.e., no retransmission happens for a message. This is reasonable because it shows the network has recovered from congestion or bloated queue.

If some retransmission has been made before the acknowledged arrives in the `FAST_SLOW_FAST` state, the sender updates the `SlowRTO` value, and moves to the third state, `SLOW_FAST`. The retransmission interval in the `SLOW_FAST` state is defined as `SlowRTO`, `FastRTO`,  $\text{FastRTO} * 2^1$ , ...,  $2^i * \text{FastRTO}$ .

In `SLOW_FAST` state, the sender switches back to the FAST state if an unambiguous acknowledgement arrives. Otherwise, the sender stays in the `SLOW_FAST` state if retransmission happens again.

#### 4.4. Retransmission Count Option

When retransmissions are needed to deliver a CoAP message, it is not possible to measure RTT for the RTO computation as the RTT sample becomes ambiguous. Therefore, it would be beneficial to be able to distinguish whether an acknowledgement arrives for the original transmission of the message or for a retransmission of it. This would allow reliably acquiring an RTT sample for every CoAP message exchange and thereby compute a more accurate RTO even during periods of congestion and loss.

The Retransmission Count Option is used to distinguish whether an Acknowledgement message arrives for the original transmission or one of the retransmissions of a Confirmable message. However, the Retransmission Count Option cannot be used with an Empty Acknowledgement (or Reset) message because the CoAP protocol specification [RFC7252] does not allow adding options to an Empty message. Therefore, Retransmission Count Option is useful only for the common case of Piggybacked Response. In case of Empty Acknowledgements the operation of FASOR is the same as without the option.

No.	C	U	N	R	Name	Format	Length	Default
TBD			X		Rexmit-Cnt	uint	0-1	0

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 1: Retransmission Count Option

Implementation of the Retransmission Count option is optional and it is identified as elective. However, when it is present in a CoAP message and a CoAP endpoint processes it, it MUST be processed as described in this document. The Retransmission Count option MUST NOT occur more than once in a single message.

The value of the Retransmission Count option is a variable-size (0 to 1 byte) unsigned integer. The default value for the option is the number 0 and it is represented with an empty option value (a zero-length sequence of bytes). However, when a client intends to use Retransmit Count option, it MUST reserve space for it by limiting the request message size also when the value is empty in order to fit the full-sized option into retransmissions.

The Retransmission Count option can be present in both the request and response message. When the option is present in a request it

indicates the ordinal number of the transmission for the request message.

If the server supports (implements) the Retransmission Count option and the option is present in a request, the server MUST echo the option value in its Piggybacked Response unmodified. If the server replies with an Empty Acknowledgement the server MUST silently ignore the option and MUST NOT include it in a later separate response to that request.

When Piggybacked Response carrying the Retransmission Count option arrives, the client uses the option to match the response message to the corresponding transmission of the request. In order to measure a correct RTT, the client must store the timestamp for the original transmission of the request as well as the timestamp for each retransmission, if any, of the request. The resulting RTT sample is used for the RTO computation. If the client retransmitted the request without the option but the response includes the option, the client MUST silently ignore the option.

The original transmission of a request is indicated with the number 0, except when sending the first request to a new destination endpoint. The first original transmission of the request to a new endpoint carries the number 255 (0xFF) and is interpreted the same as an original transmission carrying the number 0. Retransmissions, if any, carry the ordinal number of the retransmission. Once the first Piggybacked Response from the new endpoint arrives the client learns whether or not the other endpoint implements the option. If the first response includes the echoed option, the client learns that the other endpoint supports the option and may continue including the option to each retransmitted request. From this point on the original transmissions of requests implicitly include the option number 0 and a zero-byte integer will be sent according to the CoAP uint-encoding rules. If the first Piggybacked Response does not include the option, the client SHOULD stop including the option into the requests to that endpoint.

When the Retransmission Count option is in use, the client bases the retransmission timeout for the normal RTO in the back off series as follows:

max(RTO, Previous-RTT-Sample)

Previous-RTT-Sample is the RTT sample acquired from the previous message exchange. If no RTT sample was available with the previous message exchange (e.g., the server replied with an Empty Acknowledgement), RTO computed earlier is used like in case the Retransmission Count option is not in use.



#### 4.5. Alternatives for Exchanging Retransmission Count Information

An alternative way of exchanging the retransmission count information between a client and server is to encode it in the Token. The Token is a client-local identifier and a client solely decides how it generates the Token. Therefore, including a varying Token value to retransmissions of the same request is all possible as long as the client can use the Token to differentiate between requests and match a response to the corresponding request. The server is required to make no assumptions about the content or structure of a Token and always echo the Token unmodified in its response.

How exactly a client encodes the retransmission count into a Token is an implementation issue. Note that the original transmission of a request may carry a zero-length Token given that the rules for generating a Token as specified in RFC 7252 [RFC7252] are followed. This allows reducing the overhead of including the Token into the requests in such cases where Token could otherwise be omitted. However, similar to Retransmit Count option the maximum request message size MUST be limited to accommodate the Token with retransmit count into the retransmissions of the request.

#### 5. Security Considerations

#### 6. IANA Considerations

This memo includes no request to IANA.

#### 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>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [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>>.

## 7.2. Informative References

- [I-D.ietf-core-cocoa]  
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,  
"CoAP Simple Congestion Control/Advanced", draft-ietf-  
core-cocoa-03 (work in progress), February 2018.
- [JRCK18a] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "Is  
CoAP Congestion Safe?", Applied Networking Research  
Workshop (ANRW'18), July 2018.
- [JRCK18b] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "FASOR  
Retransmission Timeout and Congestion Control Mechanism  
for CoAP?", Proceedings of IEEE Global Communications  
Conference (Globecom 2018), to appear, December 2018.
- [KP87] Karn, P. and C. Partridge, "Improving Round-trip Time  
Estimates in Reliable Transport Protocols", SIGCOMM'87  
Proceedings of the ACM Workshop on Frontiers in Computer  
Communications Technology, August 1987.
- [RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks",  
RFC 896, DOI 10.17487/RFC0896, January 1984,  
<<https://www.rfc-editor.org/info/rfc896>>.

## Appendix A. Pseudocode for Basic FASOR without Dithering

```
var state = NORMAL_RTO

rfc6298_init(var fastrto, 2 secs)

var slowrto
SLOWRTO_FACTOR = 1.5

var original_sendtime
var retransmit_count

/*
 * Sending Original Copy and Retransmitting 'req'
 */
send_request(req) {
    original_sendtime = time.now
    retransmit_count = 0

    arm_rto(calculate_rto())
    send(req)
}

rto_for(req) {
    retransmit_count += 1

    arm_rto(calculate_rto())
    send(req)
}

/*
 * ACK Processings
 */
ack() {
    sample = time.now - original_sendtime
    if (retransmit_count == 0)
        unambiguous_ack(sample)
    else
        ambiguous_ack(sample)
}

unambiguous_ack(sample) {
    k = 4 // RFC6298 default K = 4
    if (rfc6298_is_first_sample(fastrto))
        k = 1
    rfc6298_update(fastrto, k, sample) // Normal RFC6298 processing
    state = NORMAL_RTO
}
```

```
ambiguous_nextstate = {
  [NORMAL_RTO] = FAST_SLOW_FAST_RTO,
  [FAST_SLOW_FAST_RTO] = SLOW_FAST_RTO,
  [SLOW_FAST_RTO] = SLOW_FAST_RTO
}

ambiguous_ack(sample) {
  slowrto = sample * SLOWRTO_FACTOR
  state = ambiguous_nextstate[state]
}

/*
 * RTO Calculations
 */
calculate_rto() {
  return <state>_rtoseries()
}

normal_rtoseries() {
  switch (retransmit_count) {
    case 0: return fastrto_series_init()
    default: return fastrto_series_backoff()
  }
}

fastslowfast_rtoseries() {
  switch (retransmit_count) {
    case 0: return fastrto_series_init()
    case 1: return MAX(slowrto, 2*fastrto)
    default: return fastrto_series_backoff()
  }
}

slowfast_rtoseries() {
  switch (retransmit_count) {
    case 0: return slowrto
    case 1: return fastrto_series_init()
    default: return fastrto_series_backoff()
  }
}

var backoff_series_timer

fastrto_series_init() {
  backoff_series_timer = fastrto
  return backoff_series_timer
}
```

```
fastrto_series_backoff() {  
    backoff_series_timer *= 2  
    return backoff_series_timer  
}
```

Figure 2

## Authors' Addresses

Ilpo Jarvinen  
University of Helsinki  
P.O. Box 68  
FI-00014 UNIVERSITY OF HELSINKI  
Finland

Email: [ilpo.jarvinen@helsinki.fi](mailto:ilpo.jarvinen@helsinki.fi)

Markku Kojo  
University of Helsinki  
P.O. Box 68  
FI-00014 UNIVERSITY OF HELSINKI  
Finland

Email: [markku.kojo@cs.helsinki.fi](mailto:markku.kojo@cs.helsinki.fi)

Iivo Raitahila  
University of Helsinki  
P.O. Box 68  
FI-00014 UNIVERSITY OF HELSINKI  
Finland

Email: [iivo.raitahila@helsinki.fi](mailto:iivo.raitahila@helsinki.fi)

Zhen Cao  
Huawei  
Beijing  
China

Email: [zhencao.ietf@gmail.com](mailto:zhencao.ietf@gmail.com)

Internet Draft  
Intended status: Standards Track  
Expires: April 2019

Joong H. Jung  
Dong K. Choi  
Seok J. Koh  
Kyungpook National University  
October 22, 2018

CoAP Sensor Streaming Using Buffer Control  
draft-jhjung-core-sensor-streaming-00.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 22, 2019.

Copyright Notice

Copyright (c) 2018 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.

## Abstract

As the Internet of Things (IoT) technology grows with the development of wireless communication and sensors, many people are interested in Constrained Application Protocol (CoAP), which is the representative protocol of the IoT. In addition, attempts have been made to apply CoAP to sensors that support existing real-time streaming services. However, the CoAP is not suitable for services to support streaming services such as smart band and CCTV. To overcome this drawback, there is an extension called CoAP Observe, but streaming services using CoAP Observe imposes a load on the server, which is not suitable for environments where low power devices act as servers, such as data transfer between sensors and gateways. In this specification, we are considering the situation in which the sensor acts as a server, and in this environment, we define one mechanism to provide efficient streaming service.

## Table of Contents

1. Introduction .....	2
1.1. Background .....	2
1.2. Terminology .....	2
1.3. Overview of the proposed scheme .....	2
2. The Buffer-Control Option .....	7
2.1. Buffer-Control Option meaning in request .....	7
2.2. Buffer-Control Option meaning in a notification .....	8
3. Subject Side Operation .....	8
3.1. Register .....	8
3.2. Caching .....	8
3.3. Change the buffer size .....	9
3.4. Unregister .....	9
4. Observer Side Operation .....	9
4.1. Register .....	9
4.2. Buffer Control .....	9
4.3. Reordering .....	9
5. Security consideration .....	10
6. IANA Considerations .....	10
7. References .....	10
7.1. Normative References .....	10
Authors' Addresses .....	10

## 1. Introduction

### 1.1. Background

The Constrained Application Protocol (CoAP) is the most widely used protocol in constrained environments such as sensor networks. Along with the growth of the Internet of Things services (IoT), a variety of real-time streaming services (e.g., CCTV) are provided by using the CoAP.

However, how to effectively use CoAP for real-time streaming services is still under study. Since the CoAP has been basically designed to support RESTful services, it may not be suitable to use for streaming services. To overcome these drawbacks, the CoAP Observe Extension [RFC7641] has been studied. This extension supports the well-known CoAP observer design pattern. But, In case that a sensor acts as a server, the extension needs to be enhanced, because a lot of loads may be given to the server.

In this document, the CoAP Observe extension [RFC7641] will be extended to support real-time streaming services so as to deal with packet error appropriately and to reduce the load on the sensor. To achieve this goal, a new option "Buffer-Control" of the CoAP Observe extension is additionally defined. This option is used to control the buffer in sensor for real-time streaming service.

### 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.

### 1.3. Overview of the proposed scheme

The main purpose of this document is to propose a new mechanism for real-time streaming. This mechanism will be effective when the sensor device periodically transmits the sensing data to the gateway, such as CCTV or smart band. In this document, a sensor device acts as a CoAP server, and a gateway acts as a client that requests data transmission to the sensor. Our description will be based on the terms defined in RFC 7641. The entity that transmits data periodically is called the subject, and the entity that receives the data is called the observer.



The receiver who wants streaming service MUST go through the registration procedure. The receiver sends a POST message to request the creation of a streaming service to the subject. Various parameters for the streaming service (e.g., authentication information, the interval between messages, or the buffer size) are transmitted with the POST message. The subject receiving the message checks the parameters and returns a 4.xx error code with an error message if it cannot create a resource for streaming. When the resource is successfully created, the subject returns the URL of the generated resource. The observer then issues a GET request with the Observe Option to the received URL, and if the resource representative is normally received, the registration procedure for the streaming service is completed. Figure 1 shows an example flow of registration process.

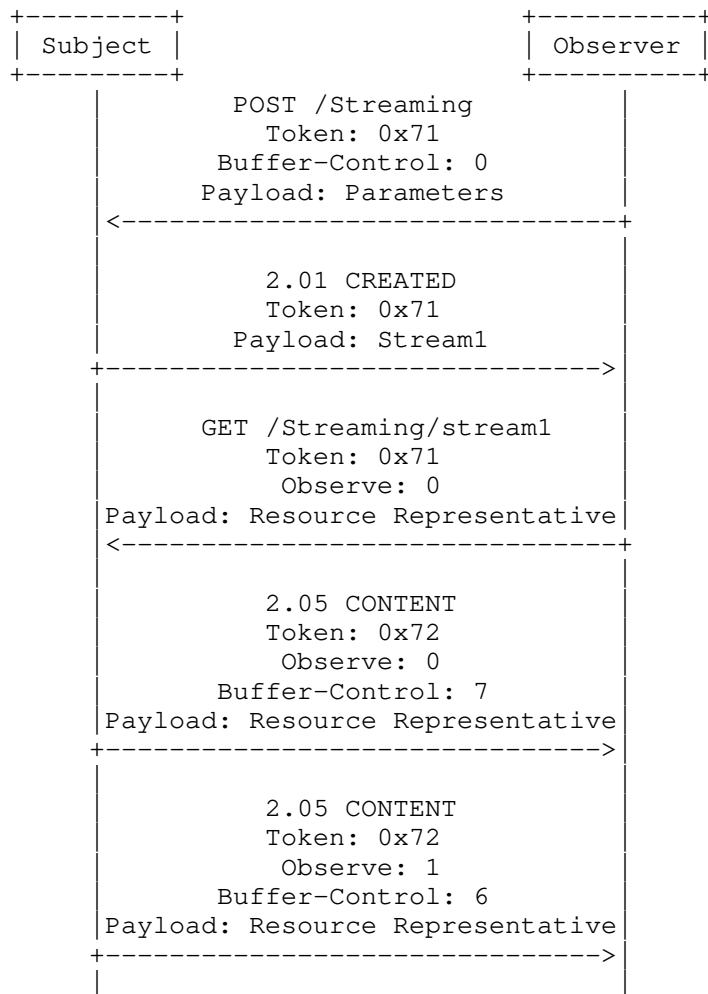


Figure 1: Streaming Service Request

When the registration process is completed, the subject periodically transmits the sensing information to the observer, until it receives a DELETE message or until the buffer is full. The subject transmits only non-confirmable messages and stores them in the buffer. If an observer receives the message, it performs reordering with Observe option value as a sequence number. Then, it transmits a PUT message to clear the buffer to avoid that the sensor buffer is full. A PUT message is transmitted for each number of messages of 'sensor buffer

size / 2 - 1'. For instance, if the buffer size is 8, when the observer receives 3 messages from the subject, it transmits a PUT message. Therefore, two PUT messages can be transmitted before the buffer is full, and even if one PUT message is lost, the transmission of the message continues without block time. The PUT message MUST include Buffer-Control option. The last sequence number received as option value until now The PUT message MUST be confirmable. The Subject receiving the PUT message deletes messages from the buffer which have a smaller sequence number than the sequence number included in the message. In other words, the PUT message acts as a cumulative ACK of TCP.

Figure 2 shows a simple example of transferring data. If a message sent by a subject is lost, the observer can send a GET request to request a message of a specific sequence number. The GET request MUST include Buffer-Control and the positive integer number meaning the specific sequence number as. The observer can also stop the streaming service by sending a DELETE message, and the subject deletes the URL when it receives a delete.

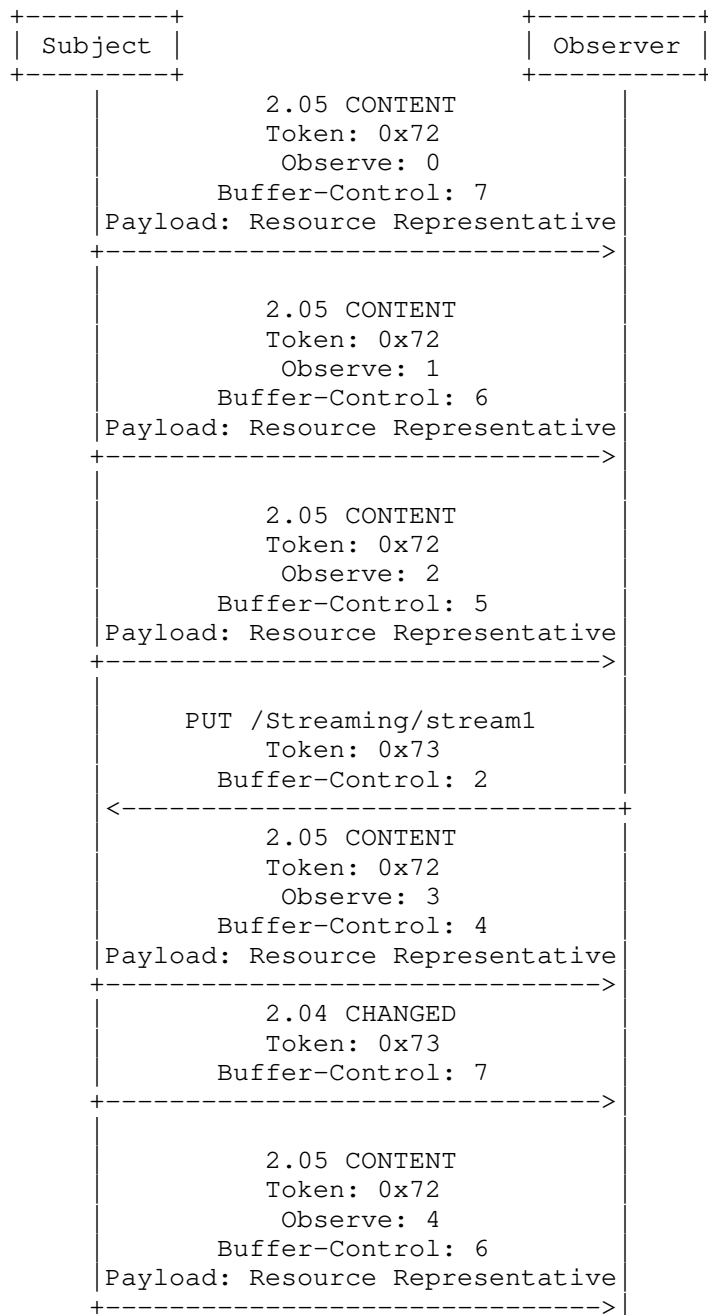


Figure 2: Transferring Data Example

## 2. The Buffer-Control Option

The Buffer-Control option has different meanings, depending on the request's method, Option Value, whether it is included in the request for the message, or included in the response.

### 2.1. Buffer-Control Option meaning in request

Table 1 shows the meaning of the options depending on the method.

	Option Value	Description
GET	Positive Integer	Sequence number of message to request
PUT	0	Buffer size to change
PUT	Positive Integer	Sequence number of the last message
POST	0	Buffer size to change

TABLE 1: Meaning of Buffer-Control Option

When included in a GET request, the Buffer-Control Option identifies the specific sequence number of the message. A GET request with the Buffer-Control option is a message that the observer transmits to the subject to request a message that was lost during transmission. In this case, the Buffer-Control option indicates the sequence number of the message to be re-requested, and the option value is a positive integer.

When the option is included in a PUT request, the Buffer-Control option has a different meaning depending on the Option value as follows :

If the option value is 0, the Buffer-Control option is used to change the buffer size. The buffer size to be changed is included in the payload.

If the option value is a positive integer number, the Buffer-Control option is used to empty the subject's buffer. In this

case, the option value includes the sequence number of the recently received message, and the Buffer-Control option is used like TCP's cumulative ack. The subject that receives the message containing this option deletes messages whose sequence number is less than or equal to the sequence number contained in the option value of the Buffer-Control option in its buffer.

The Buffer-Control option is included in the PUT message and has the same meaning as when the option value is 0. In POST messages, the Buffer-Control option has an option value of only 0.

## 2.2. Buffer-Control Option meaning in a notification

When the Buffer-Control option is included in a notification sent from the subject to the observer, that option indicates the size of the buffer that is left over.

## 3. Subject Side Operation

### 3.1. Register

The subject who supports streaming service MUST have the resource to make a resource for streaming service. In this specification, the URL of the resource is "/streaming". Also subject MUST be able to handle CoAP Observe Option.

### 3.2. Caching

The Subject MUST have a buffer to support Error Control. The buffer size is determined according to the parameters included in the POST message when the POST message for creating the resource for the streaming service is received.

### 3.3. Change the buffer size

The Subject MUST have a buffer for error control and be able to change its buffer size. The buffer size is very important in this mechanism. If the size of the buffer is large, the number of PUT messages issued to empty the buffer can be reduced. In addition, the subject maintains the message transmission until the buffer is exhausted without considering other factors, so the observer can perform congestion control by changing the buffer size of the subject. The buffer size is transmitted until the buffer is full and the transmitted message is stored in the buffer. Through the PUT message containing the Buffer-Control option received from the Observer, the messages confirmed to arrive well are deleted from the buffer.

### 3.4. Unregister

Since this mechanism basically applies between the sensor and the gateway, the resources created for the streaming service can be observed by only one Observer. Therefore, when a DELETE message is received from the Observer, the corresponding resource should be deleted and the buffer allocated for the streaming service should be released.

## 4. Observer Side Operation

### 4.1. Register

An observer who wants streaming service MUST request the creation of a resource for receiving the streaming service. In the extension defined in this specification, a resource for streaming is generated and serviced. Therefore, the observer MUST generate a streaming resource by transmitting a POST message to the subject containing various parameters such as the data resource to be streamed, the buffer size, and the interval between messages. When the resource is successfully created, it receives the resource URL to be streamed from the subject.

### 4.2. Buffer Control

For the stability of streaming services, the observer should remove the received message from the subject's buffer. This mechanism minimizes the burden on the subject as much as possible, so the observer MUST be able to manage both the error control and the state of the subject buffer. In addition, it MUST be able to change the subject's buffer size based on network conditions. For example, if the error rate is high, it is possible to control the error by transmitting the PUT message more frequently by reducing the buffer size. If the error rate is low, the buffer size can be increased to reduce the number of control messages issued.

### 4.3. Reordering

Since the CoAP message is basically UDP-based, the transmission order and reception order of messages may be different. Therefore, the observer MUST be able to re-order the message using the sequence number included in the CoAP Observe Option.

## 5. Security consideration

The security consideration will apply to Section 11 of [RFC7252], the CoAP specification, and Section 7 of [RFC7641], Observing resources in the CoAP.

## 6. IANA Considerations

TBD

## 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>>.
- [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>>.

### Authors' Addresses

Joong-Hwa Jung  
Kyungpook National University  
Daehakro 80, Bukgu, Daegu, South Korea 41566

Email: [godopul6@gmail.com](mailto:godopul6@gmail.com)



Dong-Kyu Choi  
Kyungpook National University  
Daehakro 80, Bukgu, Daegu, South Korea 41566

Email: [supergint@gmail.com](mailto:supergint@gmail.com)

Seok-Joo Koh  
Kyungpook National University  
Daehakro 80, Bukgu, Daegu, South Korea 41566

Phone: +82 53 950 7356  
Email: [sjkoh@knu.ac.kr](mailto:sjkoh@knu.ac.kr)



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 13, 2019

A. Keranen  
Ericsson  
C. Bormann  
Universitaet Bremen TZI  
March 12, 2019

SenML Data Value Content-Format Indication  
draft-keranen-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 September 13, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. SenML Content-Format ("ct") Field . . . . .	3
4. SenML Content-Type ("content-type") and Content-Coding ("content-coding") Fields . . . . .	3
5. SenML Base Content-Format ("bct"), Base Content-Type ("bcontent-type"), and Base Content-Coding ("bcontent-coding") Fields . . . . .	4
6. Security Considerations . . . . .	4
7. IANA Considerations . . . . .	4
Acknowledgements . . . . .	5
9. References . . . . .	5
9.1. Normative References . . . . .	5
9.2. Informative References . . . . .	6
Authors' Addresses . . . . .	6

## 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 decode 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 decoding 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 media 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 an unsigned integer value in the range of 0-65535 indicating the CoAP Content-Format of the data (similar to CoRE Link Format [RFC6690] "ct" attribute, except that the Link Format attribute happens to be encoded as a string).

## 4. SenML Content-Type ("content-type") and Content-Coding ("content-coding") Fields

Since some Internet media types and their content coding and parameter alternatives do not have assigned CoAP Content-Format identifiers, a Content-Type field ("content-type") can be

alternatively used to indicate the Content-Type of the data. Content-Coding of the Content-Type can be identified using a "content-coding" field. If Content-Coding is not specified with Content-Type, identity (i.e., no) transformation is used. The Content-Type field MUST NOT be used in the same record as a Content-Format field and records with Content-Format field MUST NOT contain a Content-Coding field.

#### 5. SenML Base Content-Format ("bct"), Base Content-Type ("bcontent-type"), and Base Content-Coding ("bcontent-coding") Fields

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.

The Base Content-Type Field, label "bcontent-type", provides a default value for the Content-Type Field (label "content-type") within its range. The Base Content-Coding Field, label "bcontent-coding", provides a default value for the Content-Coding Field (label "content-coding") within its range. The range and resolution rules for both are identical to Base Content-Format Field, except that the value is added with label "content-coding" (for Base Content-Coding) or with label "content-type" (for Base Content-Type).

#### 6. 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.

#### 7. IANA Considerations

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	Number	int	this document
Content-Format	ct	Number	int	this document
Base Content-Type	bcontent-type	String	string	this document
Content-Type	content-type	String	string	this document
Base Content-Coding	bcontent-coding	String	string	this document
Content-Coding	content-coding	String	string	this document

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.

#### 9. References

##### 9.1. 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>>.

## 9.2. 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-00 (work in progress), March 2019.
- [IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <<http://www.iana.org/assignments/core-parameters>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.

## Authors' Addresses

Ari Keranen  
Ericsson  
Jorvas 02420  
Finland

Email: [ari.keranen@ericsson.com](mailto:ari.keranen@ericsson.com)

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: [cabo@tzi.org](mailto:cabo@tzi.org)



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: March 21, 2019

J. Mattsson  
J. Fornehed  
G. Selander  
F. Palombini  
Ericsson  
C. Amsuess  
Energy Harvesting Solutions  
September 17, 2018

Controlling Actuators with CoAP  
draft-mattsson-core-coap-actuators-06

Abstract

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS, TLS, or OSCORE is not enough. We describe several serious attacks any on-path attacker can do, and discusses tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, some of the attacks apply equally well to sensors.

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 21, 2019.

Copyright Notice

Copyright (c) 2018 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. Attacks . . . . .	4
2.1. The Block Attack . . . . .	4
2.2. The Request Delay Attack . . . . .	5
2.3. The Response Delay and Mismatch Attack . . . . .	8
2.4. The Relay Attack . . . . .	11
2.5. The Request Fragment Rearrangement Attack . . . . .	12
2.5.1. Completing an Operation with an Earlier Final Block . . . . .	13
2.5.2. Injecting a Withheld First Block . . . . .	14
3. Security Considerations . . . . .	15
4. IANA Considerations . . . . .	15
5. References . . . . .	15
5.1. Normative References . . . . .	15
5.2. Informative References . . . . .	16
Acknowledgements . . . . .	17
Authors' Addresses . . . . .	17

## 1. Introduction

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347], TLS [RFC5246], or OSCORE [I-D.ietf-core-object-security] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC6347] and the use of CoAP over TCP and TLS is specified in [RFC8323]. OSCORE protects CoAP end-to-end with the use of COSE [RFC8152] and the CoAP Object-Security option [I-D.ietf-core-object-security], and can therefore be used over any transport.

The Constrained Application Protocol (CoAP) [RFC7252] was designed with the assumption that security could be provided on a separate

layer, in particular by using DTLS [RFC6347]. The four properties traditionally provided by security protocols are:

- o Data confidentiality
- o Data origin authentication
- o Data integrity checking
- o Replay protection

In this document we show that protecting CoAP with a security protocol on another layer is not nearly enough to securely control actuators (and in many cases sensors) and that secure operation often demands far more than the four properties traditionally provided by security protocols. We describe several serious attacks any on-path attacker (i.e. not only "trusted intermediaries) can do and discusses tougher requirements and mechanisms to mitigate the attacks. In general, secure operation of actuators also requires the three properties:

- o Data-to-Data binding
- o Data-to-space binding
- o Data-to-time binding

"Data-to-Data binding" is e.g. binding of responses to a request or binding of data fragments to each other. "Data-to-space binding" is the binding of data to an absolute or relative point in space (i.e. a location) and may in the relative case be referred to as proximity. "Data-to-time binding" is the binding of data to an absolute or relative point in time and may in the relative case be referred to as freshness. The two last properties may be bundled together as "Data-to-spacetime binding".

The request delay attack (valid for DTLS, TLS, and OSCORE and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and TLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. The request fragment rearrangement attack (valid for DTLS, TLS, and OSCORE and described in Section 2.5) lets an attacker cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations.

Mechanisms mitigating some of the attacks discussed in this document can be found in [I-D.ietf-core-echo-request-tag] and [I-D.liu-core-coap-delay-attacks]

### 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.

## 2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS, TLS, or OSCORE to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCORE, as an attacker otherwise can easily forge false requests and responses.

### 2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCORE is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies have access to the complete CoAP message, and with OSCORE, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figures 1 and 2.

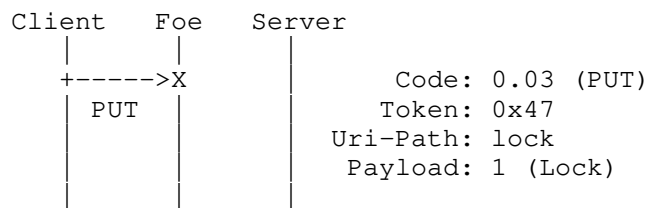


Figure 1: Blocking a request

Where 'X' means the attacker is blocking delivery of the message.

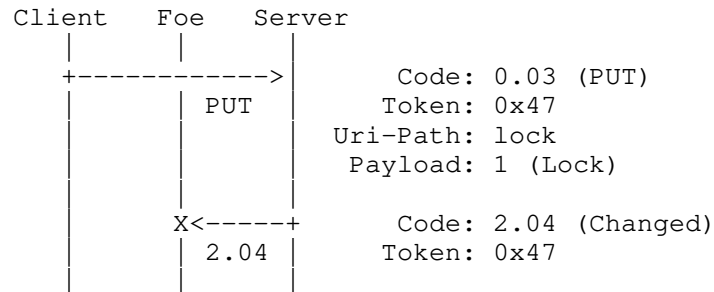


Figure 2: Blocking a response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrodinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where confirmation is important MUST notify the user upon reception of the response, or warn the user when a response is not received.

## 2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCORE, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, or OSCORE, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCORE allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks. The replay window has a default length of 64 in DTLS and 32 in OSCORE. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was

delayed and will therefore happily process the request. Note that delays can also happen for other reasons than a malicious attacker.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

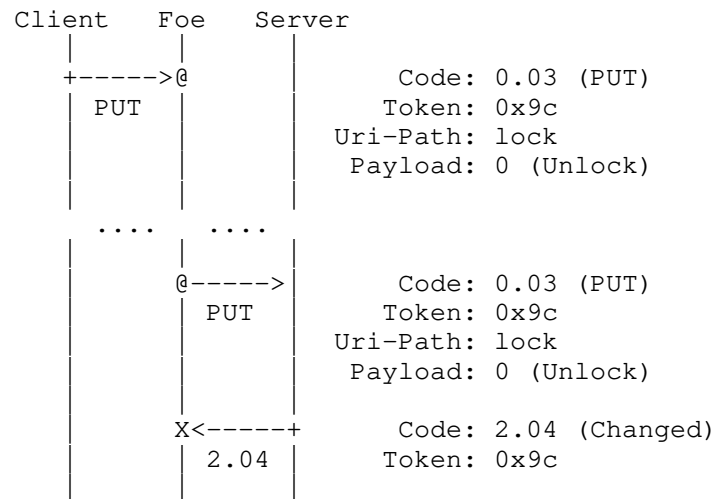


Figure 3: Delaying a request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will

have a different sequence number and the attacker can forward it. If OSCORE is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

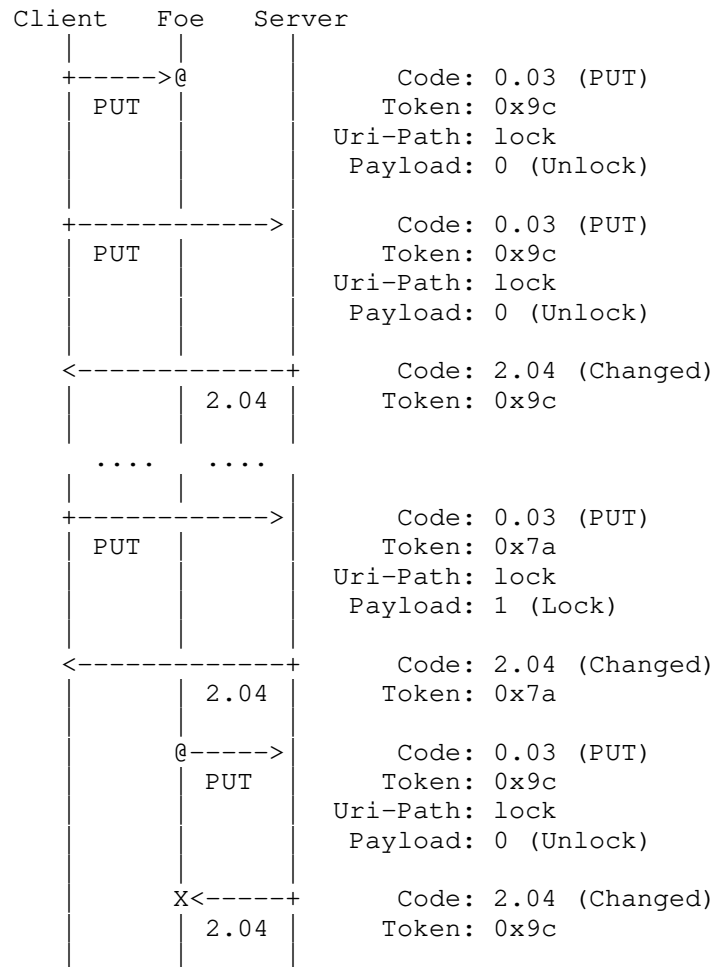


Figure 4: Delaying request with reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent or enable the server to securely determine an absolute point in time when the

request is to be executed. This can be accomplished with either a challenge-response pattern, by exchanging timestamps between client and server, or by only allowing requests a short period after client authentication.

Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange [I-D.selander-ace-cose-ecdhe]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on exchanging timestamps require exactly synchronized time between client and server, and this may be hard to control with complications such as time zones and daylight saving. Wall clock time SHOULD NOT be used as it is not monotonic, may reveal that the endpoints will accept expired certificates, or reveal the endpoint's location. Use of non-monotonic clocks is not secure as the server will accept requests if the clock is moved backward and reject requests if the clock is moved forward. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism where the server does not need to synchronize its time with the client is easier to analyze but require more roundtrips. The challenges, responses, and timestamps may be sent in a CoAP option or in the CoAP payload.

Remedy: The mechanisms specified in [I-D.ietf-core-echo-request-tag] or [I-D.liu-core-coap-delay-attacks] SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

### 2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS, TLS, and IPsec, but not OSCORE. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS and TLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token, the response will be accepted by the client as a valid response to the later request. If CoAP is used over a reliable and ordered transport such as TCP with TLS, no messages can be delivered before the delayed message. If CoAP is



used over an unreliable and unordered transport such as UDP with DTLS, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. 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.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

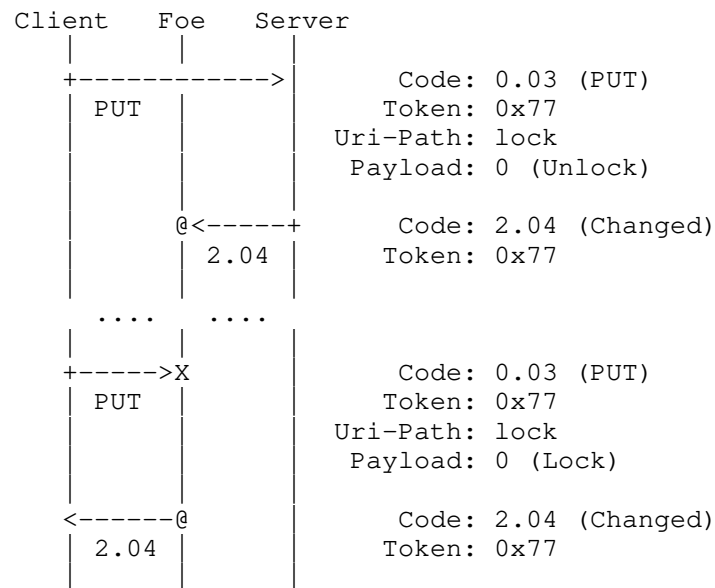


Figure 5: Delaying and mismatching response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

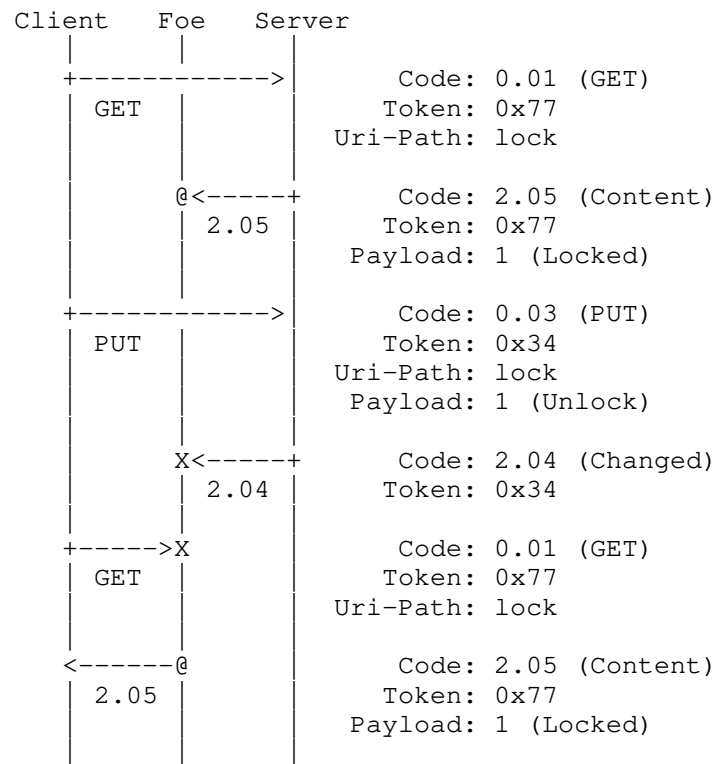


Figure 6: Delaying and mismatching response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same (D)TLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a (D)TLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

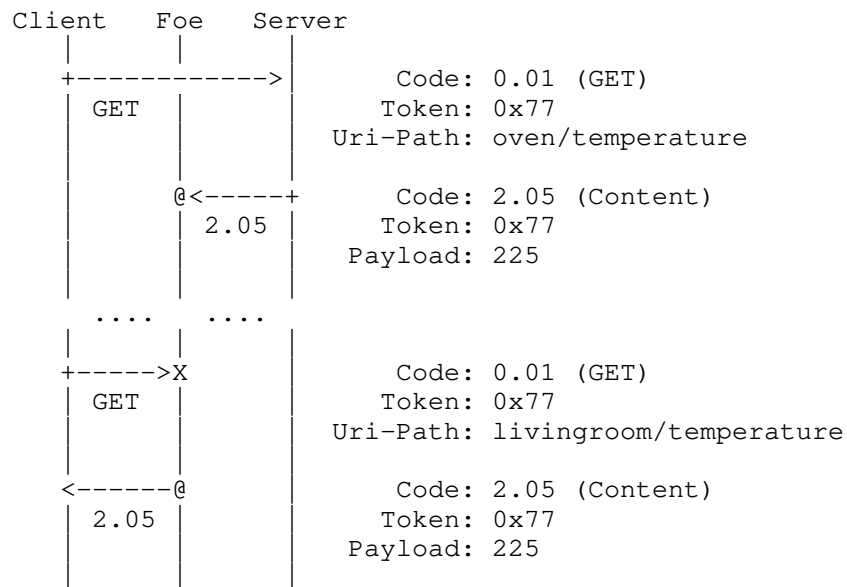


Figure 7: Delaying and mismatching response from other resource

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) the client **MUST NOT** reuse any tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach **SHOULD** be followed.

## 2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

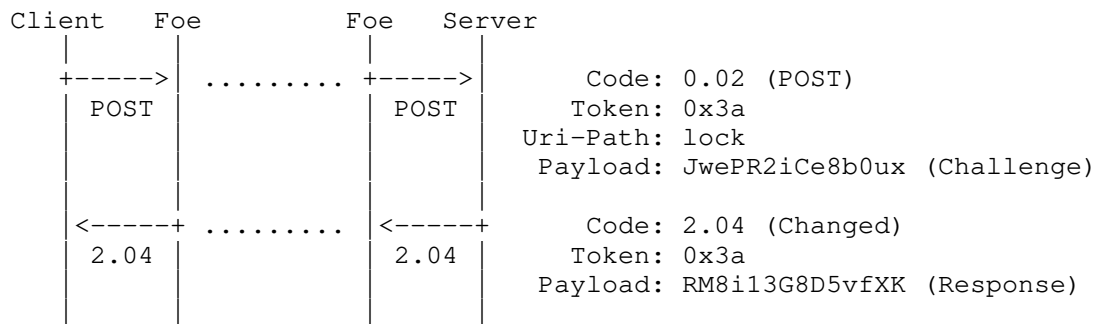


Figure 8: Relay attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters) that cannot be relayed through e.g. clothes. Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

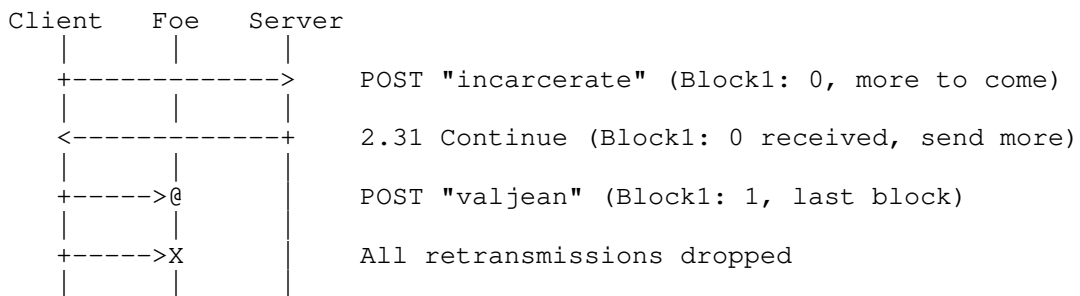
## 2.5. The Request Fragment Rearrangement Attack

These attack scenarios show that the Request Delay and Block Attacks can be used against blockwise transfers to cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations. The combination of these attacks is described as a separate attack because it makes the Request Delay Attack relevant to systems that are otherwise not time-dependent, which means that they could disregard the Request Delay Attack.

This attack works even if the individual request/response pairs are encrypted, authenticated and protected against the Response Delay and Mismatch Attack, provided the attacker is on the network path and can correctly guess which operations the respective packages belong to.

### 2.5.1. Completing an Operation with an Earlier Final Block

In this scenario (illustrated in Figure 9), blocks from two operations on a POST-accepting resource are combined to make the server execute an action that was not intended by the authorized client. This works only if the client attempts a second operation after the first operation failed (due to what the attacker made appear like a network outage) within the replay window. The client does not receive a confirmation on the second operation either, but, by the time the client acts on it, the server has already executed the unauthorized action.



(Client: Odd, but let's go on and promote Javert)

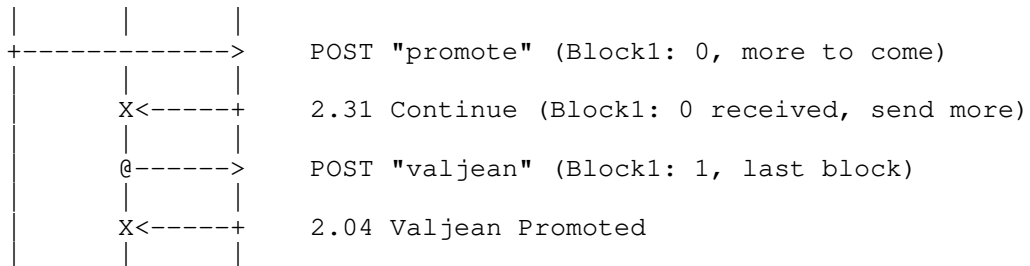


Figure 9: Completing an operation with an earlier final block

**Remedy:** If a client starts new blockwise operations on a security context that has lost packages, it needs to label the fragments in such a way that the server will not mix them up.

A mechanism to that effect is described as Request-Tag [I-D.ietf-core-echo-request-tag]. Had it been in place in the example and used for body integrity protection, the client would have set the Request-Tag option in the "promote" request. Depending on the server's capabilities and setup, either of four outcomes could have occurred:

1. The server could have processed the reinjected POST "valjean" as belonging to the original "incarcerate" block; that's the expected case when the server can handle simultaneous block transfers.
2. The server could respond 5.03 Service Unavailable, including a Max-Age option indicating how long it prefers not to take any requests that force it to overwrite the state kept for the "incarcerate" request.
3. The server could decide to drop the state kept for the "incarcerate" request's state, and process the "promote" request. The reinjected POST "valjean" will then fail with 4.08 Request Entity incomplete, indicating that the server does not have the start of the operation any more.

#### 2.5.2. Injecting a Withheld First Block

If the first block of a request is withheld by the attacker for later use, it can be used to have the server process a different request body than intended by the client. Unlike in the previous scenario, it will return a response based on that body to the client.

Again, a first operation (that would go like "Homeless stole apples. What shall we do with him?" - "Set him free.") is aborted by the proxy, and a part of that operation is later used in a different operation to prime the server for responding leniently to another operation that would originally have been "Hitman killed someone. What shall we do with him?" - "Hang him.". The attack is illustrated in Figure 10.

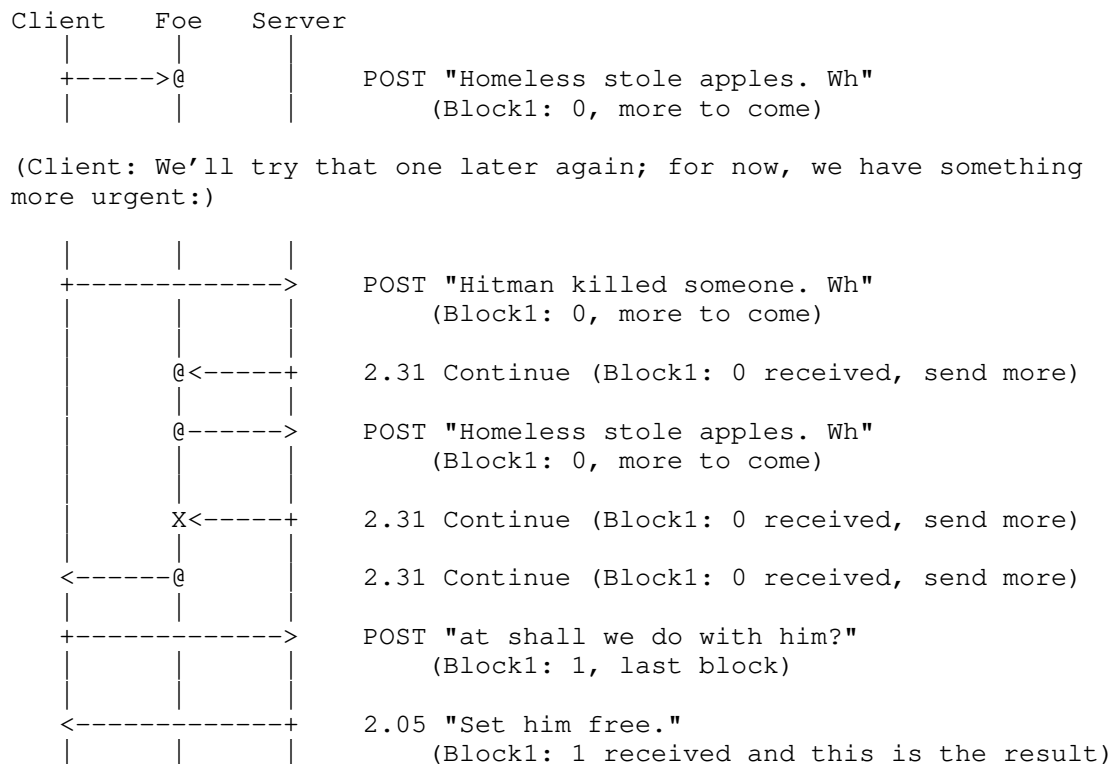


Figure 10: Injecting a withheld first block

### 3. Security Considerations

The whole document can be seen as security considerations for CoAP.

### 4. IANA Considerations

This document has no actions for IANA.

### 5. References

#### 5.1. 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>>.
- [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>>.

## 5.2. Informative References

- [I-D.ietf-core-echo-request-tag]  
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-02 (work in progress), June 2018.
- [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-15 (work in progress), August 2018.
- [I-D.liu-core-coap-delay-attacks]  
Liu, Y. and J. Zhu, "Mitigating delay attacks on Constrained Application Protocol", draft-liu-core-coap-delay-attacks-01 (work in progress), October 2017.
- [I-D.selander-ace-cose-ecdhe]  
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-09 (work in progress), July 2018.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.



[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>>.

#### Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Ari Keraenen, Matthias Kovatsch, Sandeep Kumar, and Andras Mehes for their valuable comments and feedback.

#### Authors' Addresses

John Mattsson  
Ericsson AB  
SE-164 80 Stockholm  
Sweden

Email: [john.mattsson@ericsson.com](mailto:john.mattsson@ericsson.com)

John Fornehed  
Ericsson AB  
SE-164 80 Stockholm  
Sweden

Email: [john.fornehed@ericsson.com](mailto:john.fornehed@ericsson.com)

Goeran Selander  
Ericsson AB  
SE-164 80 Stockholm  
Sweden

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

Francesca Palombini  
Ericsson AB  
SE-164 80 Stockholm  
Sweden

Email: [francesca.palombini@ericsson.com](mailto:francesca.palombini@ericsson.com)

Christian Amsuess  
Energy Harvesting Solutions

Email: [c.amsuess@energyharvesting.at](mailto:c.amsuess@energyharvesting.at)

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

M. Tiloca  
RISE AB  
C. Amsuess

P. van der Stok  
Consultant  
March 11, 2019

Discovery of OSCORE Groups with the CoRE Resource Directory  
draft-tiloca-core-oscore-discovery-02

Abstract

Group communication over the Constrained Application Protocol (CoAP) can be secured by means of Object Security for Constrained RESTful Environments (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 CoAP endpoints can use the CoRE Resource Directory to discover OSCORE groups and acquire information to join them through their respective Group Manager. A same 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 September 12, 2019.

## Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	2
1.1. Terminology . . . . .	4
2. Registration Resource for Group Managers . . . . .	5
3. Registration of Group Manager Endpoints . . . . .	5
4. Addition and Update of OSCORE Groups . . . . .	6
5. Discovery of OSCORE Groups . . . . .	7
5.1. Discovery Example #1 . . . . .	8
5.2. Discovery Example #2 . . . . .	9
6. Security Considerations . . . . .	10
7. IANA Considerations . . . . .	10
7.1. Resource Types . . . . .	10
8. References . . . . .	11
8.1. Normative References . . . . .	11
8.2. Informative References . . . . .	11
Acknowledgments . . . . .	12
Authors' Addresses . . . . .	12

## 1. Introduction

A set of CoAP endpoints may share a common pool of resources, hence composing an application group. All the members of an application group may also be members of a same security group, hence sharing a common set of keying material to secure group communication.

The Constrained Application Protocol (CoAP) [RFC7252] supports group communication over IP multicast [RFC7390] to improve efficiency and latency of communication and reduce bandwidth requirements. The method Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] enables 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 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 the responsible 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 as ACE Client and Resource Server, respectively. That is, the joining endpoint accesses the join resource associated to the OSCORE group of interest 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 join 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: the OSCORE group(s) to join and the responsible GM(s) are generally unknown at manufacturing time; 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; 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 updated information about those groups.

This specification describes how CoAP endpoints can use the CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory] for discovering an OSCORE group and retrieving the information required to join that group through the responsible 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 join resource.

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 convenient notifications about the creation of new OSCORE groups or the updates of information concerning existing ones. Thus, it facilitates the early deployment of CoAP endpoints, i.e. even before the GM is deployed and the OSCORE groups of interest 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 a same OSCORE Common Security Context to protect group communication as described in [I-D.ietf-core-oscore-groupcomm]. That is, 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 to a single OSCORE group, while different application groups can rely on the same OSCORE group. Application groups MAY share resources. Any two application groups associated to the same OSCORE group do not share any resource.
- o Zeroed-epoch Group ID: this refers to the Group ID of an OSCORE group as stored in the RD. The structure of such a stored Group

ID is as per Appendix C of [I-D.ietf-core-oscore-groupcomm], with the "Group Epoch" part immutable and set to zero.

## 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 join resources at the GM, associated to the OSCORE groups under that GM.

In particular, each link to a join resource includes:

- o "target": URI of the join resource at the GM.
- o target attributes, including:
  - \* Resource Type (rt) with the value "core.osc.j" defined in Section 7.1 of this specification.
  - \* The zeroed-epoch Group ID of the OSCORE group.
  - \* One target attribute for each application group associated to the OSCORE group, specifying the name of that application group.

## 3. Registration of Group Manager Endpoints

Upon deployment, a GM finds the RD as described in Section 4 of [I-D.ietf-core-resource-directory]. After that, the GM registers as an endpoint with the RD, as described in Section 5.3 of [I-D.ietf-core-resource-directory].

When doing so, the GM MUST also register all the join resources it is exporting at that point in time, i.e. one for each of its OSCORE groups.

For each registered join resource, the GM MUST specify the following parameters in the payload of the registration request.

- o 'rt' = "core.osc.j" (see Section 7.1).
- o 'oscore-gid', specifying the zeroed-epoch Group ID 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. 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.

The GM SHOULD NOT use the Simple Registration approach described in Section 5.3.1 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 link to one join resource for accessing the OSCORE group with zeroed-epoch Group ID "feedca570000" and used by one application group with name "group1".

Request: GM -> RD

```
Req: POST coap://rd.example.com/rd?ep=gm1
Content-Format: 40
Payload:
</join/feedca570000>;ct=41;rt="core.osc.j";
oscore-gid="feedca570000";app-gp="group1"
```

Response: RD -> GM

```
Res: 2.01 Created
Location-Path: /rd/4521
```

#### 4. Addition and Update of OSCORE Groups

The GM is responsible to keep its registration with the RD up to date with links to all its join resources. This means that the GM has to update the registration within its lifetime as per Section 5.4.1 of [I-D.ietf-core-resource-directory], and has to change the content of the registration when a join 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 join resource.
- o The GM dismisses an OSCORE group and stops exporting the related join resource.
- o Information related to an existing OSCORE group changes, e.g. the list of associated application groups.

In order to perform an update to the set of links in its registration, the GM can re-register with the RD and fully specify all links to its join resources and their target attributes in the payload of the POST request.



The example below shows the same GM from Section 3 that re-registers with the RD. When doing so, it specifies:

- o The same previous join resource associated to the OSCORE group with zeroed-epoch Group ID "feedca570000".
- o A second join resource associated to the OSCORE group with zeroed-epoch Group ID "ech0ech00000" and used by one application group, namely "group2".
- o A third join resource associated to the OSCORE group with zeroed-epoch Group ID "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:

```
</join/feedca570000>;ct=41;rt="core.osc.j";  
oscore-gid="feedca570000";app-gp="group1",  
</join/ech0ech00000>;ct=41;rt="core.osc.j";  
oscore-gid="ech0ech00000";app-gp="group2",  
</join/abcdef120000>;ct=41;rt="core.osc.j";  
oscore-gid="abcdef120000";app-gp="group3";app-gp="group4"
```

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 5.4.3 of [I-D.ietf-core-resource-directory]. This requires semantics to be defined in future standards, in order to apply a link-format document as a patch to a different one.

## 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], in order 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 MUST consider the following search criteria for the lookup filtering.

- o 'rt' = "core.osc.j" (see Section 7.1).

The joining node MAY additionally consider the following search criteria for the lookup filtering, depending on the information it has already available.

- o 'oscore-gid', specifying the zeroed-epoch Group ID of the OSCORE group of interest. This parameter MUST specify a single value.
- o 'ep', specifying the identifier of the GM as endpoint registered with the RD.
- o 'app-gp', specifying the name(s) of the application group(s) associated to 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.

#### 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 to the application group "group1", but that does not know the zeroed-epoch Group ID of the OSCORE group, the responsible GM and the join resource to access.

Request: Joining node -> RD

Req: GET coap://rd.example.com/lookup/res?rt=core.osc.j&app-gp=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;rt="core.osc.j";  
oscore-gid="feedca570000";app-gp="group1";  
anchor="coap://[2001:db8::ab]"
```

If it does not know the multicast IP address used in "group1", the joining node can retrieve it by performing 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/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 zeroed-epoch Group ID "feedca570000", but that does not know the responsible GM, the join resource to access, and the associated application groups.

The example also shows how the joining node uses observation [RFC7641], in order to be notified of possible changes in the join resource's target attributes. 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 available at a later point in time.

Request: Joining node -> RD

Req: GET coap://rd.example.com/lookup/res?rt=osc.j&\  
oscore-gid=feedca570000  
Observe: 0

Response: RD -> Joining node

Res: 2.05 Content

Observe: 24

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;rt="osc.j";  
oscore-gid="feedca570000";app-gp="group1";  
anchor="coap://[2001:db8::ab]"
```

Depending on the used search criteria, the joining node performing the resource lookup can get a response whose payload is quite large in size. This can happen, for instance, in case the lookup request targets all the join resources at a specified GM, or all the join resources of all the registered GMs, as in the example below.

Request: Joining node -> RD

Req: GET coap://rd.example.com/lookup/res?rt=osc.j

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;rt="osc.j";
oscore-gid="feedca570000";app-gp="group1";
anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/join/ech0ech00000>;rt="osc.j";
oscore-gid="ech0ech00000";app-gp="group2";
anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::cd]/join/abcdef120000>;rt="osc.j";
oscore-gid="abcdef120000";app-gp="group3";app-gp="group4";
anchor="coap://[2001:db8::cd]"
```

Therefore, it is RECOMMENDED that a joining node performing a resource lookup to discover OSCORE groups uses observation only when including the fine-grained search criterion 'oscore-gid' in its GET request sent to the RD.

## 6. Security Considerations

The security considerations as described in Section 8 of [I-D.ietf-core-resource-directory] apply here as well.

## 7. IANA Considerations

This document has the following actions for IANA.

### 7.1. 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.j	Join resource of an OSCORE Group Manager	[[this document]]

## 8. References

### 8.1. Normative 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-01 (work in progress), March 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-04 (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-19 (work in progress), January 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>>.
- [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-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-22 (work in progress), March 2019.

- [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.
- [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>>.
- [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, Francesca Palombini 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](mailto:marco.tiloca@ri.se)

Christian Amsuess  
Hollandstr. 12/4  
Vienna 1020  
Austria

Email: [christian@amsuess.com](mailto:christian@amsuess.com)

Peter van der Stok  
Consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)  
Email: [consultancy@vanderstok.org](mailto:consultancy@vanderstok.org)  
URI: [www.vanderstok.org](http://www.vanderstok.org)

March 4, 2019

Expires: September 2019

Blockchain Transaction Protocol for Constraint Nodes  
draft-urien-core-blockchain-transaction-protocol-02.txt

## Abstract

The goal of the blockchain transaction protocol for constraint nodes is to enable the generation of blockchain transactions by constraint nodes, according to the following principles :

- transactions are triggered by Provisioning-Messages that include the needed blockchain parameters.
- binary encoded transactions are returned in Transaction-Messages, which include sensors/actuators data. Constraint nodes, associated with blockchain addresses, compute the transaction signature.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 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 September 2019.

.



## Copyright Notice

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

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

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	4
2 Overview of the Blockchain Transaction Protocol for Constraint Nodes.....	4
2.1 Architecture.....	4
2.2 An Ethereum Use Case.....	5
3 Blockchain Transaction Protocol Messages Definition.....	6
3.1 Provisioning Message.....	6
3.1.1 Encoding example in JSON syntax .....	6
3.2 Transaction Message.....	6
3.2.1 Encoding example in JSON syntax .....	6
4. Blockchain Transaction Protocol Messages Binary Encoding.....	7
4.1 CoAP messages.....	7
4.2 HTTP Messages.....	7
5 IANA Considerations.....	7
6 Security Considerations.....	7
6 References.....	7
6.1 Normative References.....	7
6.2 Informative References.....	7
7 Authors' Addresses.....	7

## 1 Overview

In the context of this draft sensors/actuators are powered by micro-controllers comprising about 10KB of RAM and 100KB of non volatile memory. The node electronic board may include a radio SoC (System On Chip) or the micro-controller can be part of the SoC. The radio chip manages IP connectivity with another device, typically acting as a controller, which provides a full internet access with standard computing resources.

A constraint node driving sensors and/or actuators may deliver critical data dealing with safety (fire detection,...) or legacy (pollution measurement,...) information.

Blockchain infrastructure provides two important features in an Internet of Things (IoT) context:

- Authentication of data in P2P context. Blockchain signed transactions are checked by numerous nodes.
- Information publication. Transactions are stored in duplicated and distributed databases.
- Dating information. Transactions are dated during the mining process.

The goal of the blockchain transaction protocol for constraint nodes is to enable the generation of blockchain transactions by constraint nodes, according to the following principles:

- transactions are triggered by controllers. Needed blockchain parameters are included in provisioning messages.
- binary encoded transaction messages are returned by constraint nodes. A node has the ability to compute the transaction signature.

## 2 Overview of the Blockchain Transaction Protocol for Constraint Nodes

### 2.1 Architecture

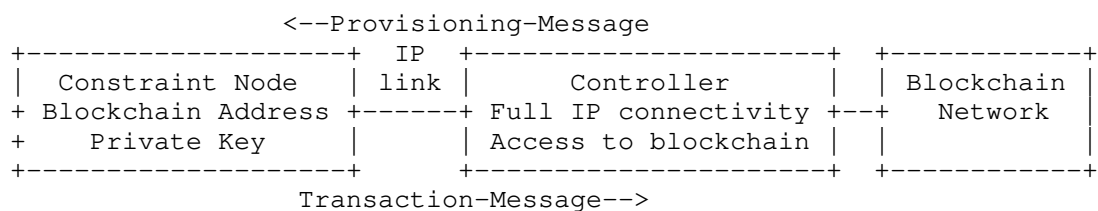


Figure 1. Functional architecture for the Blockchain Transaction Protocol for Constraint Nodes

A constraint node holds a blockchain address (BA). The blockchain address is computed from a private key (Pk). Most of today blockchain infrastructures deal with ECDSA signatures, generated

over the Secp256k1 elliptic curve. The private key is a 32 bytes number, stored in the constraint node. The computation of hash procedures such a SHA2 or KECCAK-256 can be handled by microcontrollers. Although ECDSA signature may be generated by a microcontroller, a tamper resistant resource could be used, either embedded in the CPU, or in a chip such as a secure element[ISO7816]. As an illustration an architecture based on micro-controller, radio SoC and secure element was demonstrated in [IEEE-CCNC2018].

The controller is a device with full IP connectivity. It typically communicates with the constraint node thanks to the CoAP [RFC7252] protocol, or other legacy protocols such as HTTPS. The controller has access to the blockchain infrastructure, to which it is able to forward a binary encoded transaction, signed by the constraint node.

## 2.2 An Ethereum Use Case.

The following figure 2, illustrates an Ethereum transaction generated by a constraint node, whose total length is 118 bytes.

```
F8 74 // RLP List, length= 116 bytes
0C // nonce 1 byte =12 decimal
85 06FC23AC00 // gasPrice = 30 GWei
83 013880 // gasLimit = 80000 gas
// recipient address 20 bytes
94 6BAC1B75185D9051AF740AB909F81C71BBB221A6
80 // Null Ether Value
// Data 15 bytes "Temperature=25C"
8F 54656D70657261747572653D323543
1B // recovery parameter, 1 byte
A0 // r, 32 bytes, ECDSA r parameter
A9B58980F76EE6284800B82A2B5DF13E456887EC0CF426A5E5D6A738EB1784ED
A0 // s, 32 bytes, ECDSA s parameter
629633C6A3ED5FEE0FB40E2D1CF251345B885D372857B1A6C4762C9BE914281F
```

Figure 2. Illustration of an Ethereum transaction, generated by a constraint node.

The identifier (TxId) of this transaction (i.e. its KECCAK-256 digest) is:

```
0xd6904d832462ae17718c69e9caa0c3f3bed458382ac1f4e43b1aadd8e94744ad
```

Given this TxId, the transaction can be retrieved in any Ethereum blockchain database, like for example:

```
https://etherscan.io/tx/0xd6904d832462ae17718c69e9caa0c3f3bed458382ac1f4e43b1aadd8e94744ad
```

The transaction date (20-2018 09:52:42 PM +UTC) is published and certified by the blockchain.

The binary encoded transaction comprises two parts,

- information relying on the Ethereum blockchain context, such as the nonce, the gasPrice, the gasLimit, the recipient address, and an Ether value.
- information delivered by the constraint node, data (a temperature measurement), and the ECDSA signature computed from the 32 bytes private key.

Parameters relying on the Ethereum blockchain context MUST be included in the Provisioning-Message.  
The signed transaction MUST be included in the Transaction-Message.

### 3 Blockchain Transaction Protocol Messages Definition

The Blockchain Transaction Protocol comprises two messages, to be included in transport protocols, such as CoAP or HTTP.

#### 3.1 Provisioning Message

This message includes the following attributes :

- A type, an integer value, specifying the message content.
- An ordered list of values, storing the parameters of the blockchain context.

##### 3.1.1 Encoding example in JSON syntax

Here is an illustration of the provisioning message associated to the Ethereum blockchain.

```
{
  "type": 1,
  "nonce": 12,
  "gasPrice": 30,
  "gasLimit": 80000,
  "address": "6BAC1B75185D9051AF740AB909F81C71BBB221A6",
  "value": 0
}
```

#### 3.2 Transaction Message

This message include the following attributes

- A type, an integer value, specifying the message content. The zero value indicates an error.
- The binary encoded transaction, including the signature.

##### 3.2.1 Encoding example in JSON syntax

Here is an illustration of the transaction message associated to the Ethereum blockchain.

```
{
  "type": 1,
  "transaction":
    "F8740C8506FC23AC0083013880946BAC1B75185D9051AF740AB909F81C71BBB221A
    6808F54656D70657261747572653D3235431BA0A9B58980F76EE6284800B82A2B5DF
    13E456887EC0CF426A5E5D6A738EB1784EDA0629633C6A3ED5FEE0FB40E2D1CF2513
    45B885D372857B1A6C4762C9BE914281F"
}
```

#### 4. Blockchain Transaction Protocol Messages Binary Encoding

##### 4.1 CoAP messages

To be Done

##### 4.2 HTTP Messages

To be Done

#### 5 IANA Considerations

TODO

#### 6 Security Considerations

TODO

#### 6 References

##### 6.1 Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

##### 6.2 Informative References

[IEEE-CCNC2018] Urien, P., "An Innovative Security Architecture for Low Cost Low Power IoT Devices Based on Secure Elements", IEEE CCNC 2018

#### 7 Authors' Addresses

Pascal Urien  
Telecom ParisTech  
23 avenue d'Italie  
75013 Paris  
France

Phone: NA  
Email: Pascal.Urien@telecom-paristech.fr

Urien

Expires September 2019

[Page 7]

December 2018

Expires: June 2019

Identity Modules for CoAP  
draft-urien-core-identity-module-coap-05.txt

## Abstract

This document defines identity modules based on Secure Elements processing DTLS/TLS stacks for CoAP devices. The expected benefits of these secure microcontrollers are the following :

- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between CoAP devices and CoAP clients.
- The device identity is enforced by a non cloneable chip.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 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 June 2019.

.

## Copyright Notice

Copyright (c) 2018 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

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	4
2 What is a Secure Element.....	4
3 Identity Module for CoAP.....	6
4 DTLS/TLS profile for CoAP security modules.....	6
5 IANA Considerations.....	6
6 References.....	7
6.1 Normative References.....	7
6.2 Informative References.....	7
7 Authors' Addresses.....	7

## 1 Overview

The CoAP [CoAP] protocol MAY be secured by the DTLS protocol [DTLS] over an UDP/IP stack; the TLS support [TLS] is also under definition [CoAP-TLS] over a TCP/IP stack.

According to [CoAP] four security modes are available, NoSec, PreSharedKey, RawPublicKey, and Certificate. When DTLS is used with the PreShareKey or Certificate modes there is a need to store secrets such as symmetric or asymmetric keys, which authenticate the CoAP device.

In that case a Secure Element (SE) MAY be used in order to fully run the DTLS or TLS protocol. According to the data throughput or other security considerations the DTLS/TLS session MAY be exported from the secure element after the exchange of the finished messages.

This class of Secure Element is referred by this draft as an identity module (IdMod).

The expected benefits of identity modules are the following :

- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between the CoAP device and the CoAP client.
- The device identity is enforced by a non cloneable identity module.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

## 2 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller (see figure 1) equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 5x5 mm<sup>2</sup>. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), non volatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control (PKCS15 cards).

Most of secure elements include a Java Virtual Machine (JVM) and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security

purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical low cost Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

According to the state of art, TLS/DTLS stacks may run in secure elements, for example written as a javacard applications.

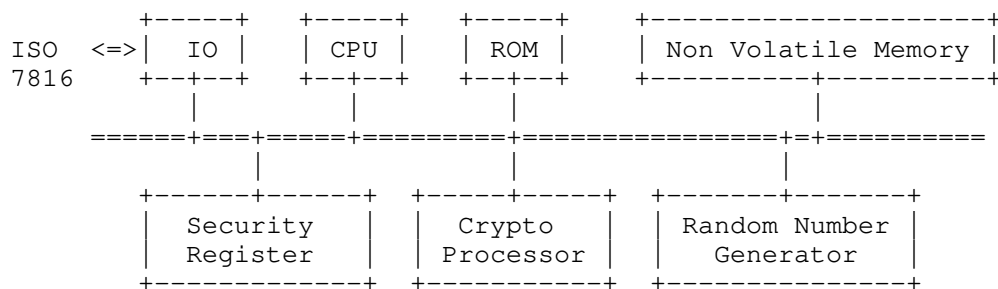


Figure 1. A typical hardware architecture of a Secure Element

## 3 Identity Module for CoAP

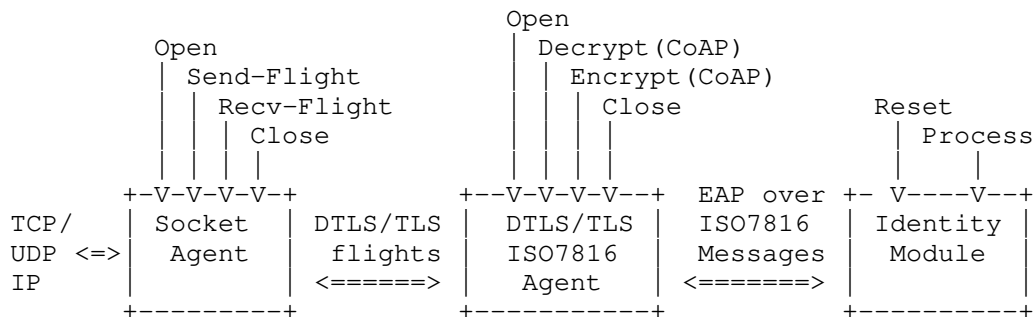


Figure 2. CoAP Identity module framework

ISO7816 interface for Secure Elements providing TLS/DTLS stacks are detailed in [DTLS/TLS-SM]. The Identity module MUST support two commands Reset and Process.

TLS/DTLS packets are transported by the EAP protocol over ISO7816 messages. This mechanism previously detailed by [EAPSC] provides a double segmentation procedure thanks to EAP and ISO7816 facilities.

A DTLS/TLS-ISO7816 software agent sends and receives DTLS/TLS flights to/from sockets over EAP/ISO7816 messages to/from the identity module. Conceptually this component interface SHOULD have four procedures Open, Close, Encrypt and Decrypt.

A socket software agent extracts and send DTLS/TLS flights from/to UDP/TCP packets. Conceptually this component interface SHOULD have four procedures Open, Close, Recv-Flight, Send-Flight.

## 4 DTLS/TLS profile for CoAP security modules

To be done.

## 5 IANA Considerations

This draft does not require any action from IANA.

## 6 References

### 6.1 Normative References

[TLS] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 5746, August 2008

[DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[CoAP-TLS] A TCP and TLS Transport for the Constrained Application Protocol (CoAP), draft-ietf-core-coap-tcp-tls-02, April 2016.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

### 6.2 Informative References

[GP] Global Platform Standards, <http://www.globalplatform.org>

[EUROSMART] The EUROSMART association, <http://www.eurosmart.com>

[DTLS/TLS-SM] Urien, P., "TLS and DTLS Security Modules", draft-urien-uta-tls-dtls-security-module-05.txt, December 2017

[EAPSC] Urien, P., "EAP Support in Smartcard", draft-urien-eap-smartcard-34.txt, December 2017

## 7 Authors' Addresses

Pascal Urien  
Telecom ParisTech  
23 avenue d'Italie  
75013 Paris  
France

Phone: NA  
Email: [Pascal.Urien@telecom-paristech.fr](mailto:Pascal.Urien@telecom-paristech.fr)

CORE Working Group  
Internet Draft  
Intended status: Experimental

P. Urien  
Telecom ParisTech

December 2018

Expires: June 2019

Remote APDU Call Secure (RACS)  
draft-urien-core-racs-12.txt

## Abstract

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm<sup>2</sup>; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements. It is designed according to the representational State Transfer (REST) architecture. RACS resources are identified by dedicated URIs. An HTTP interface is also supported.

An open implementation [OPENRACS] is available  
(<https://github.com/purien>) for various OS.

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 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 June 2019.

.

Urien

Expires June 2019

[Page 1]

## Copyright Notice

Copyright (c) 2018 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

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	5
1.1 What is a Secure Element.....	5
1.2 Grid Of Secure Elements (GoSE).....	6
1.3 Secure Element Identifier (SEID).....	7
1.3.1 SlotID example .....	7
1.3.2 SEID for Secure Elements .....	8
1.4 APDUs.....	9
1.4.1 ISO7816 APDU request .....	9
1.4.2 ISO7816 APDU response .....	9
2 The RACS protocol.....	10
2.1 Structure of RACS request.....	10
2.2 Structure of a RACS response.....	11
2.2.1 BEGIN Header .....	11
2.2.2 END Header .....	11
2.2.3 Status line .....	11
2.2.4 Examples of RACS responses: .....	12
2.3 RACS request commands.....	12
2.3.1 BEGIN .....	12
2.3.2 END .....	12
2.3.3 The APPEND parameter .....	13
2.3.4 GET-VERSION .....	14
2.3.5 SET-VERSION .....	14
2.3.6 LIST .....	15
2.3.7 RESET .....	15
2.3.8 APDU .....	16
2.3.9 SHUTDOWN .....	19
2.3.10 POWERON .....	20
2.3.11 ECHO .....	21
2.4 Status header encoding.....	21
2.4.1 Event class .....	22
2.4.2 Command class .....	22
3 URI for the GoSE.....	23
4 HTTP interface.....	23
4.1 HTTPS Request.....	23
4.2 HTTPS response.....	24
5 Security Considerations.....	24
5.1 Authorization.....	24
5.2 Secure Element access.....	24
5.3 Applications security policy.....	25
5.3.1 Users-Table .....	25
5.3.2 SEID-Table .....	25
5.3.3 APDU-Table .....	25
5.4 Overview of the security policy.....	26
6 IANA Considerations.....	26
7 References.....	26
7.1 Normative References.....	26



7.2 Informative References.....	26
8 Authors' Addresses.....	27

## 1 Overview

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grids of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm<sup>2</sup>; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements.

RACS is designed according to the representational State Transfer (REST) architecture [REST], which encompasses the following features:

- Client-Server architecture.
- Stateless interaction.
- Cache operation on the client side.
- Uniform interface.
- Layered system.
- Code On Demand.

### 1.1 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 25mm<sup>2</sup>. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), nonvolatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control.

Most of secure elements include a Java Virtual Machine and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

## 1.2 Grid Of Secure Elements (GoSE)

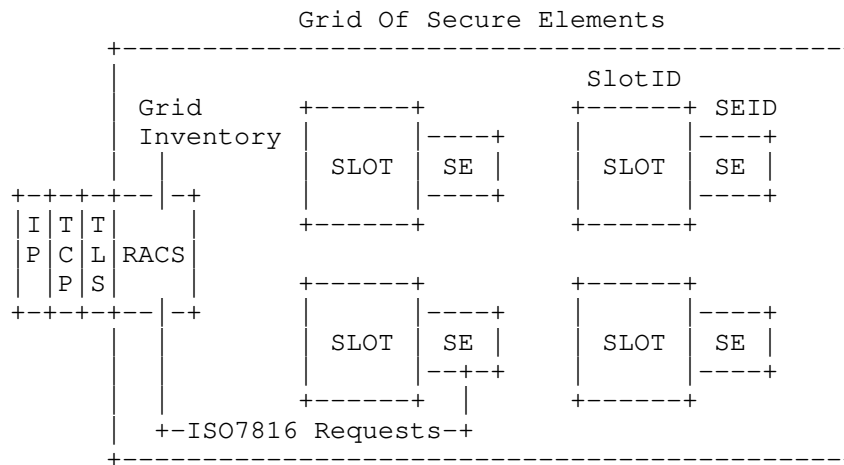


Figure 1. Architecture of a Grid of Secure Elements

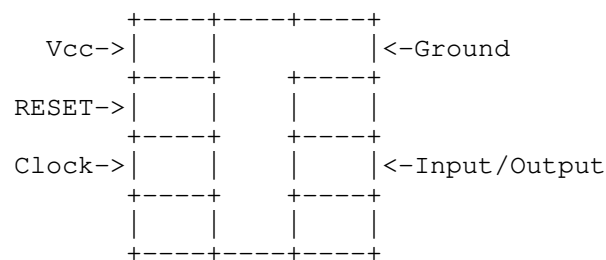


Figure 2. Illustration of an ISO7816 Secure Element

A grid of Secure Elements (GoSE) is a server hosting a set of secure elements.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes,

- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

A grid of Secure Elements offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

According to this draft all accesses to a GoSE require the TCP transport and are secured by the TLS [TLS 1.0] [TLS 1.1] [TLS 2.0] protocol.

The RACS protocol provides all the features needed for the remote use of secure elements, i.e.

- Inventory of secure elements
- Information exchange with the secure elements

### 1.3 Secure Element Identifier (SEID)

Every secure element needs a physical slot that provides electrical feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device, i.e. a reader) supporting host buses such as USB.

Within the GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name.

The SEID (Secure Element Identifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs.

Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

#### 1.3.1 SlotID example

According to the PC/SC (Personal Computer/Smart Card) standard [PS/SC], a smart card reader MAY include a serial number. This attribute (VENDOR-IFD-SERIAL) is associated to the tag 0x0103 in the class VENDOR-INFO.

### 1.3.2 SEID for Secure Elements

According to the Global Platform standard [GP] the Issuer Security Domain (ISD) manages applications lifecycle (downloading, activation, deletion). The command 'initialize update' is used to start a mutual authentication between the administration entity and the secure element; it collects a set of data whose first ten bytes are called the 'key diversification data'. This information is used to compute symmetric keys, and according for example to [EMV] MAY comprise a serial number.

#### 1.4 APDUs

According to the [ISO7816] standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDUs (application protocol data unit).

##### 1.4.1 ISO7816 APDU request

An APDU request comprises two parts: a header and an optional body.

The header is a set of four or five bytes noted CLA INS P1 P2 P3

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
- INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1 P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures)
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes
- The body may be empty. Its maximum size is 255 bytes

##### 1.4.2 ISO7816 APDU response

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.
- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

An error free operation is usually associated to the 9000 status word. Following are some interpretations of the tuple SW1, SW2 according to various standards:

- '61' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '9F' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '6C' 'XX', the P3 value is wrong, request must be performed again with the LE parameter value sets to 'XX'
- '6E' 'XX', wrong instruction class (CLA) given in the request
- '6D' 'XX', unknown instruction code (INS) given in the request
- '6B' 'XX', incorrect parameter P1 or P2
- '67' 'XX', incorrect parameter P3
- '6F' 'XX', technical problem, not implemented...

## 2 The RACS protocol

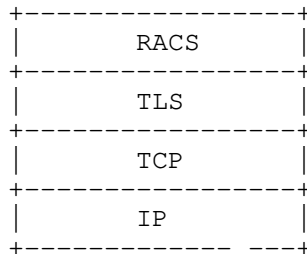


Figure 2. The RACS stack

The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. The TLS client (i.e. the RACS client) MUST be authenticated by a certificate.

One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests towards a secure element in order to perform cryptographic operations in the user's plane. In that case a RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.

The mandatory use of TLS with mutual authentication based on certificate provides a simple and elegant way to establish the credentials of a RACS client over the GoSE. It also enables an easy splitting between users' and administrators' privileges.

### 2.1 Structure of RACS request

A RACS request is a set of command lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each command is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the command to be executed.

A command line MAY comprise other tokens, which are called the command parameters.

A RACS request MUST start by a BEGIN command and MUST end by an END command.

Each command line is associated to an implicit line number. The BEGIN line is associated to the zero line number.

The processing of a RACS request is stopped after the first error. In that case the returned response contained the error status induced by the last executed command.

## 2.2 Structure of a RACS response

A RACS response is a set of lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each line is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the header.

The second token of response each line is associated command line number

A response line MAY comprise other tokens, which are called the response parameters.

Three classes of headers are defined BEGIN, END and Status.

A RACS response MUST start by a BEGIN header and MUST end by an END header. It comprises one or several status lines.

### 2.2.1 BEGIN Header

This header starts a response message.

It comprises an optional parameter, an identifier associated to a previous request message.

### 2.2.2 END Header

This header ends a response message.

### 2.2.3 Status line

A status header indicates a status line.

It begins by the character '+' in case of success or '-' if an error occurred during the RACS request execution. It is followed by an ASCII encoded integer, which is the value of the status.

The second mandatory token of a status line is the command line number (starting from zero)



A status line MAY comprise other tokens, which are called the response parameters.

#### 2.2.4 Examples of RACS responses:

```
BEGIN CrLf
+001 000 Success CrLf
END CrLf
```

```
BEGIN moon1969 CrLf
-301 007 Illegal command, BEGIN condition not satisfied at line 7
END CrLf
```

```
BEGIN Asterix237 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf
```

```
BEGIN CrLf
-100 002 Unknown command at line 2 CrLf
END CrLf
```

```
BEGIN CrLf
-606 001 Unauthorized command APDU command at line 1
END CrLf
```

```
BEGIN CrLf
-706 001 SEID Already in use, APDU command at line 1
END CrLf
```

### 2.3 RACS request commands

#### 2.3.1 BEGIN

This command starts a request message. A response message is returned if an error is detected.

An optional parameter is the request identifier, which MUST be echoed in the parameter of the first response line (i.e. starting by the BEGIN header).

#### 2.3.2 END

This command ends a request message. It returns the response message triggered by the last command.

## Example1

=====

## Request:

BEGIN CrLf

END CrLf

## Response:

BEGIN CrLf

+001 000 Success CrLf

END CrLf

## Example2

=====

## Request:

BEGIN Marignan1515 CrLf

APDU ASTERIX-CRYPTO-MODULE [ISO7816-Request] CrLf

END CrLf

## Response:

BEGIN Marignan1515 CrLf

+006 001 [ISO7816-Response] CrLf

END CrLf

## 2.3.3 The APPEND parameter

The APPEND parameter MAY be used in all command lines, excepted BEGIN and END. The APPEND parameter MUST be the last parameter of a command line.

By default a response message returns only the last status line.

When APPEND is inserted, the command line, if executed, MUST produce a status line.

## Example

## Request:

BEGIN SanchoPanza CrLf

APDU 100 [ISO7816-Request-1] CrLf

APDU 100 [ISO7816-Request-2] CrLf

END CrLf

## Response:

BEGIN SanchoPanza CrLf

+006 002 [ISO7816-Response-2] CrLf

END CrLf

## Request:

BEGIN DonQuichotte CrLf

APDU 100 [ISO7816-Request-1] APPEND CrLf

APDU 100 [ISO7816-Request-2] APPEND CrLf

END CrLf

Response:  
BEGIN DonQuichotte CrLf  
+006 001 [ISO7816-Response-1] CrLf  
+006 002 [ISO7816-Response-2] CrLf  
END CrLf

#### 2.3.4 GET-VERSION

This command requests the current version of the RACS protocol. The returned response is the current version encoded by two integer separated by the '.' character. The first integer indicates the major version and the second integer gives the minor version.

This draft version is 0.2

##### Example

=====

Request:  
BEGIN CrLf  
GET-VERSION CrLf  
END CrLf

Response:  
BEGIN CrLf  
+002 001 1.0 CrLf  
END CrLf

#### 2.3.5 SET-VERSION

This command sets the version to be used for the RACS request. An error status is returned by the response if an error occurred.

##### Example 1

=====

Request:  
BEGIN CrLf  
SET-VERSION 2.0 CrLf  
END CrLf

Response:  
BEGIN CrLf  
-403 001 Error line 1 RACS 2.0 is not supported CrLf  
END CrLf

##### Example 2

=====

Request:  
BEGIN CrLf  
SET-VERSION 1.0 CrLf  
END CrLf

```
Response:
BEGIN CrLf
+003 001 RACS 1.0 has been activated CrLf
END CrLf
```

### 2.3.6 LIST

This command requests the list of SEID plugged in the GoSE.

It returns a list of SEIDs separated by space (0x20) character(s).

Some SEID attributes MAY be built from a prefix and an integer suffix (such as SE#100 in which SE# is the suffix and 100 is the integer suffix. A list of non-consecutive SEID MAY be encoded as prefix[i1;i2;..;ip] where i1,i2,ip indicates the integer suffix. A list of consecutive SEID could be encoded as prefix[i1-ip] where i1,i2,ip indicates the integer suffix.

#### Example 1

=====

```
Request:
BEGIN CrLf
LIST CrLf
END CrLf
```

```
Response:
BEGIN CrLf
+004 001 SEID1 SEID2 CR LF
END CrLf
```

#### Example 2

=====

```
Request:
BEGIN CrLf
LIST CrLf
END CrLf
```

```
Response:
BEGIN CrLf
+004 001 Device[1000-2000] SerialNumber[567;789;243] CrLf
END CrLf
```

### 2.3.7 RESET

This command resets a secure element. The first parameter gives the secure element identifier (SEID). An optional second parameter specifies a warm reset. The default behavior is a cold reset. The response status indicates the success or the failure of this operation.

Syntax: RESET SEID [WARM] CrLf

Example 1

=====

Request:

BEGIN CrLf

RESET device#45 CrLf

END CrLf

Response:

BEGIN CrLf

+005 001 device#45 Reset Done

END CrLf

Example 2

=====

Request:

BEGIN CrLf

RESET device#45 CrLf

END CrLf

Response:

BEGIN CrLf

-705 001 error device#45 is already in use

END CrLf

Example 3

=====

Request:

BEGIN CrLf

RESET device#45 WARM CrLf

END CrLf

Response:

BEGIN CrLf

+005 001 device#45 Warm Reset Done CrLf

END CrLf

### 2.3.8 APDU

This command sends an ISO7816 request to a secure element or a set of ISO7816 commands.

The first parameter specifies the SEID.

The second parameter is an ISO7816 request.

Three optional parameters are available; they MUST be located after the second parameter.

- CONTINUE=value, indicates that the next RACS command will be executed only if the ISO7816 status word (SW) is equal to a given value. Otherwise an error status is returned.
- MORE=value, indicates that a FETCH request will be performed (i.e. a new ISO7816 request will be sent) if the first byte of the ISO7816 status word (SW1) is equal to a given value.
- FETCH=value fixes the four bytes of the ISO7816 FETCH request (i.e. CLA INS P1 P2). The default value (when FETCH is omitted) is 00C00000 (CLA=00, INS=C0, P1=00, P2=00)

When the options CONTINUE and MORE are simultaneously set the SW1 byte is first checked. If there is no match then the SW word is afterwards checked.

The ISO7816 6Cxx status MUST be autonomously processed by the GoSE.

#### SYNTAX

APDU SEID ISO7816-REQUEST [CONTINUE=SW] [MORE=SW1] [FETCH=CMD] CrLf

The returned response is the ISO7816 response. If multiple ISO7816 requests are executed (due to the MORE option), the bodies are concatenated in the response, which ends by the last ISO7816 status word.

The pseudo code of the APDU command is the following :

```

1. BODY = empty;
2. SW   = empty;
3. DoIt = true;
3. Do
4. { iso7816-response = send(iso7816-request);
5.   body || sw1 || sw2 = iso7816-response;
6.   If ( (first request) && (iso7816-request.size==5) &&
        (body==empty) && (sw1==6C) )
7.   { iso7816-request.P3 = sw2 ; }
6.   Else
7.   { SW = sw1 || sw2
8.     BODY = BODY || body;
9.     If (sw1 == MORE)
10.    { iso7816-request = FETCH || sw2 ; }
11.    Else
12.    { DoIt=false;}
13.  }
14. }
15. While (DoIt == true)

16. iso7816-response = BODY || SW ;
17. If (SW != CONTINUE) Error ;
18. Else
    No Error;
```

## Example 1

=====

## Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

## Response:

BEGIN CrLf

+006 001 ISO7816-RESPONSE CrLf

END CrLf

## Example 2

=====

## Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

## Response:

BEGIN CrLf

-706 001 error SEID is already used CrLf

END CrLf

## Example 3

=====

## Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

## Response:

BEGIN CrLf

-606 001 error access unauthorized access CrLf

END CrLf

## Example 4

=====

BEGIN CrLf

APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf

APDU SEID ISO7816-REQUEST-2 CrLf

END CrLf

## Response:

BEGIN CrLf

+006 002 ISO7816-RESPONSE-2 CrLf

END CrLf

## Example 5

=====

```
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf
```

## Response:

```
BEGIN CrLf
-006 001 Request Error line 1 wrong SW CrLf
END CrLf
```

## Example 6

=====

```
BEGIN CrLf
APDU SEID ISO7816-REQ-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-2 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-3 CONTINUE=9000 MORE=61 FETCH=00C00000 CrLf
END CrLf
```

## Response:

```
BEGIN CrLf
+006 003 ISO7816-RESP-3 CrLf
END CrLf
```

Multiple ISO7816 requests have been performed by the third APDU command according to the following scenario :

- the ISO7816-REQ-3 request has been forwarded to the secure element (SEID)
- the ISO 7816 response comprises a body (body-0) and a status word (SW-0) whose first byte is 0x61, and the second byte is SW2-0
- the FETCH command CLA=00, INS=00, P1=00, P2=00, P3=SW2-0 is sent to the secure element
- the ISO 7816 response comprises a body (body-1) and a status word (SW-1) set to 9000

The RACS response is set to  
+006 003 body-0 || body-1 || SW-1 CrLf  
where || indicates a concatenation operation.

## 2.3.9 SHUTDOWN

This command powers down a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: SHUTDOWN SEID CrLf



## Example

=====

## Request:

```
BEGIN Goodbye CrLf
SHUTDOWN device#45 CrLf
END CrLf
```

## Response:

```
BEGIN Goodbye CrLf
+007 001 device#45 has been powered down CrLf
END CrLf
```

## 2.3.10 POWERON

This command powers up a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: POWERON SEID CrLf

## Example 1

=====

## Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

## Response:

```
BEGIN CrLf
+008 001 device#45 Has been powered up CrLf
END CrLf
```

## Example 2

=====

## Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

## Response:

```
BEGIN CrLf
-708 001 error device#45 is already in use CrLf
END CrLf
```

## Example 3

=====

## Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

```
Response:
BEGIN CrLf
-608 001 error unauthorized access CrLf
END CrLf
```

### 2.3.11 ECHO

This command echoes a token. The first parameter is the token (word) to be echoed by the response.

Syntax: ECHO SEID CrLf

#### Example 1

=====

```
Request:
BEGIN TestEcho CrLf
ECHO Hello CrLf
END CrLf
```

```
Response:
BEGIN TestEcho CrLf
+009 001 Hello CrLf
END CrLf
```

#### Example 2

=====

```
Request:
BEGIN ResetSEID CrLf
POWERON device#45 CrLf
ECHO Done CrLf
END CrLf
```

```
Response:
BEGIN ResetSEID CrLf
+009 001 Done CrLf
END CrLf
```

## 2.4 Status header encoding

The first token of a response line is the status header. It begins by a '+' or a '-' character, and comprises three decimal digits (xyz).

The first digit (x) MUST indicate an event class.  
The second and third digits (yz) MAY indicate a command class.

#### 2.4.1 Event class

This draft only defines the meaning of the first digit located at the left most side.

- +0yz: No error
- 0yz: Command execution error
- 1yz: Unknown command, the command is not defined by this draft
- 2yz: Not implemented command
- 3yz: Illegal command, the command can't be executed
- 4yz: Not supported parameter or parameter illegal value
- 5yz: Parameter syntax error or parameter missing
- 6yz: Unauthorized command
- 7yz: Already in use, a session with this SE is already opened
- 8yz: Hardware error
- 9yz: System error

#### 2.4.2 Command class

The second and third digits (yz) MAY indicates the command that triggered the current line status

- 01 BEGIN
- 02 GET-VERSION
- 03 SET-VERSION
- 04 LIST
- 05 RESET
- 06 APDU
- 07 SHUTDOWN
- 08 POWERON
- 09 ECHO

### 3 URI for the GoSE

The URI addressing the resources hosted by the GoSE is represented by the string:

`RACS://GoSE-Name:port/?request`

where request is the RACS request to be forwarded to a the GoSE.

RACS command lines are encoded in a way similar to the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters, is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a request is written to the following syntax  
`cmd1=cmd1-parameters&cmd2=cmd2-parameters`

Example:

`RACS://GoSE-Name:port/?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=`

### 4 HTTP interface

A GoSE SHOULD support an HTTP interface. RACS requests/responses are transported by HTTP messages. The use of TLS is mandatory.

#### 4.1 HTTPS Request

`https://GoSE-Name:port/RACS?request`

where request is the RACS request to be forwarded to a secure element (SEID)

The RACS request is associated to an HTML form whose name is "RACS". The request command lines are encoded as the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a RACS request is written as  
`https://GoSE-Name/RACS?cmd1=cmd1-parameters&cmd2=cmd2-parameters`

Example:

`https://GoSE-Name/RACS?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=`

## 4.2 HTTPS response

The RACS response is returned in an XML document.

The root element of the document is <RACS-Response>

The optional parameter of the BEGIN header, is the content of the <begin> element.

Each status line is the content of the <Cmd-Response> element, which includes the following information :

- The status header is the content of the <status> element.
- The line number is the content of the <line> element.
- The other parameters of the status line are the content of the <parameters> element.

The END header is associated to the element <end>

End of line, i.e. carriage return (Cr) and line feed (Lf) characters are omitted.

As a consequence a RACS response is written as :

```
<RACS-Response>
<begin>Optionnal-ID</begin>
<Cmd-Response
<status>+000</status>
<line>001</line>
<parameters>other parameters of the RACS response</parameters>
</Cmd-Response>
<end></end>
</RACS-Response>
```

## 5 Security Considerations

### 5.1 Authorization

A RACS client MUST be authenticated by an X509 certificate.

The GoSE software MUST provide a mean to establish a list of SEIDs that can be accessed from a client whose identity is the CommonName (CN) attribute of its certificate. It MAY allocate a UserID (UID), i.e. an integer index from the certificate common name.

### 5.2 Secure Element access

The GoSE MUST manage a unique session identifier (SID) for each TLS session. The SID is bound to the client's certificate CommonName (SID(CN))

A secure element has two states, unlocked and locked. In the locked state the secure element may be only used by the SID that previously locked it.

The first authorized command that successfully accesses to a SEID (either POWERON ,RESET, APDU) locks a secure element (SEID) with the current session (SID).

The SHUTDOWN command MUST unlock a secure element (SEID).

The end of a TLS session MUST unlock all the secure elements locked by the session.

### 5.3 Applications security policy

According to the [ISO7816] standards each Application embedded within a secure element (associated to a SEID) is identified by an AID parameter (16 bytes at the most)

The RACS server SHOULD support the following facilities

#### 5.3.1 Users-Table

Each CN (the Users-Table primary key) is associated to a list of SEIDs whose access is authorized.

#### 5.3.2 SEID-Table

Each AID (the SEID-Table primary key) is associated to a list of CNs whose access is authorized.

#### 5.3.3 APDU-Table

For a given AID and an authorized CN, an APDU-Table MAY be available. This table acts as a firewall, which defined a set of forbidden ISO7816 commands.

For example this filter could be expressed as a set of the four first bytes of an APDU-Prefix (CLA INS P1 P2) and a four bytes Mask  
An ISO7816-Request is firewall if:

ISO7816-Request AND Mask IsEQUAL to APDU-Prefix

## 5.4 Overview of the security policy

The summary of the security policy is illustrated by the figure 3.

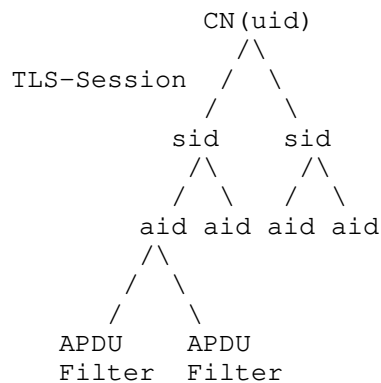


Figure 3. Summary of the security policy

## 6 IANA Considerations

This draft does not require any action from IANA.

## 7 References

### 7.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 5746, August 2008

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

### 7.2 Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

[GP] Global Platform Standards, <http://www.globalplatform.org>

[EUROSMART] The EUROSMART association, <http://www.eurosmart.com>

[PC/SC] The PC/SC workgroup, <http://www.pcscworkgroup.com>

[EMV] EMV Card Personalization Specification, Version 1.1, July 2007

[OPENRACS] <https://github.com/purien>, open RACS implementation for Win32, Ubuntu, Raspberrypi

## 8 Authors' Addresses

Pascal Urien  
Telecom ParisTech  
23 avenue d'Italie  
75013 Paris  
France

Phone: NA  
Email: [Pascal.Urien@telecom-paristech.fr](mailto:Pascal.Urien@telecom-paristech.fr)



Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: March 25, 2019

M. Veillette, Ed.  
Trilliant Networks Inc.  
September 21, 2018

Constrained YANG Module Library  
draft-veillette-core-yang-library-03

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 CoMI 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 March 25, 2019.

Copyright Notice

Copyright (c) 2018 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 . . . . .	13
8. References . . . . .	14
8.1. Normative References . . . . .	14
8.2. Informative References . . . . .	14
Author's Address . . . . .	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 [I-D.ietf-netconf-rfc7895bis] section 2 and 3.

## 2. Terminology and Notation

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

The following terms are defined in [RFC7950]: client, deviation, feature, module, submodule and server.

The following term is defined in [I-D.ietf-core-yang-cbor]: YANG Schema Item identifier (SID).

The following terms are defined in [I-D.ietf-netconf-rfc7895bis]: YANG library and YANG library checksum.

### 3. Overview

The conceptual model of the YANG library is depicted in Figure 1.

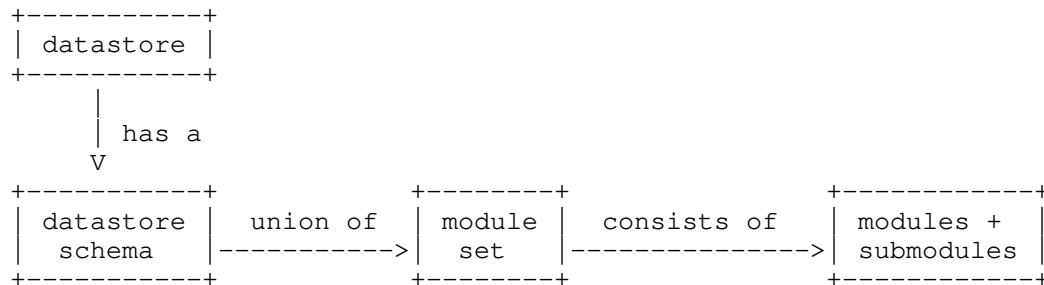


Figure 1: Abstract CoMI architecture

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 provide:

- 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    comi:sid
        +--ro revision?     revision-identifier
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    comi:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
        +--ro feature*      comi:sid
        +--ro deviation*    -> ../../module/identifier
      +--ro import-only-module* [identifier revision]
        +--ro identifier    comi:sid
        +--ro revision      union
        +--ro namespace     inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    comi: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 list of 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 doesn'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@2018-01-20.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 CoMI reference.

  import ietf-comi {
    prefix comi;
    reference "I-D.ietf-core-comi";
  }
  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 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 comi: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 comi: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 comi: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;
        mandatory true;
        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

This YANG module is designed to be accessed via the CoMI protocol [I-D.ietf-core-comi]. 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 identify server implementations with such a defect, in order to launch a denial of service attack on the device.

## 7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, ietf-yang-library [I-D.ietf-netconf-rfc7895bis], 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

- [I-D.ietf-core-yang-cbor]  
Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-07 (work in progress), September 2018.
- [I-D.ietf-netconf-rfc7895bis]  
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

### 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-03 (work in progress), June 2018.

Author's Address

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

Phone: +14503750556

Email: [michel.veillette@trilliantinc.com](mailto:michel.veillette@trilliantinc.com)

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

C. Yang, Ed.  
SY. Pan, Ed.  
South China University of Technology  
HB. Sun  
Inspur  
KM. Qu  
NetEase, Inc  
GQ. Han  
South China University of Technology  
October 23, 2018

The Standards on a Cloud Service Framework and Protocol for  
Construction, Migration, Deployment, and Publishing of Internet-Oriented  
Scalable Web Software Systems in Non-Programming Mode  
draft-yangcan-core-web-software-built-in-cloud-00

## Abstract

This draft mainly focuses on the scalable architecture and publishing protocol standard of REST-based SAAS cloud model Web software in non-programming mode, stipulates the data structure pattern and data exchange protocol for the construction and release of REST-based scalable Web cloud service software systems. Using the standardized framework and protocol, users can easily and quickly design their own software systems in the cloud, transfer and release data, which may make conventional software development so ease to improve the efficiency of complex database construction and server management. Without having to write codes under the standard framework, users can get consistent style background to create service, rapidly develop web application systems with the function of standard data management and data maintenance, and directly publish the software system to the end users of the Internet for access and use. And provide RESTful APIs to facilitate external access to required service resources. The framework can thus greatly shorten the software development life cycle, and save a great deal of development cost and maintenance overhead.

## 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 26, 2019.

#### Copyright Notice

Copyright (c) 2018 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. Definitions and Terminology . . . . .	4
3. Technical Specific Implementation Standards . . . . .	6
3.1. System Architecture Diagram . . . . .	6
3.2. SAAS Cloud Platform Related Implementation Standards . .	8
3.2.1. Application Management Module . . . . .	8
3.2.2. End User Management Module . . . . .	8
3.2.3. Relationship Management Module between the End User and the Application . . . . .	8
3.2.4. Resource Management Module . . . . .	9
3.2.5. Fine-grained Access Rights Management Module . . . .	9
3.2.6. Ways of Accessing Cloud Storage Resources . . . . .	10
3.2.7. Format of Describing the Structured Data URI . . . .	10
3.2.8. Contents of the Ways of Accessing Cloud Storage Resources . . . . .	10
3.3. Web Software Framework Related Implementation Standards .	11
3.3.1. System Administrator View . . . . .	11
3.3.2. User View . . . . .	12
3.4. Implementation Standards on Construction Methods of Web Software . . . . .	12
3.4.1. System Requirements Table and Data Record Exchange Table . . . . .	13
3.4.1.1. System Requirements Table . . . . .	13
3.4.1.2. Data Record Exchange Table . . . . .	14

3.4.2.	Steps of Reading System Requirements Table . . . . .	14
3.4.3.	Steps of Analyzing System Requirements Table . . . . .	15
3.4.4.	Contents of Modules for Users . . . . .	15
3.4.5.	Steps of Principle of Constructing Web Software System . . . . .	17
3.4.6.	Steps of Injecting Data Records into Web Software System . . . . .	19
3.4.7.	Submission of System Requirements Table . . . . .	19
3.4.8.	Submission of Data Exchange Table . . . . .	19
3.5.	Implementation Standards on Web Software Deployment . . .	20
3.5.1.	System Deployment . . . . .	20
3.5.2.	Model Deployment . . . . .	20
3.5.3.	User Deployment . . . . .	20
3.5.4.	User Group Deployment . . . . .	21
3.6.	Implementation Standards on Web Software Migration . . .	21
3.6.1.	Model Migration . . . . .	21
3.6.2.	User Migration . . . . .	21
3.6.3.	User Group Migration . . . . .	22
3.6.4.	User Group Deployment . . . . .	22
4.	Security Considerations . . . . .	22
5.	IANA Considerations . . . . .	23
6.	References . . . . .	23
6.1.	Normative References . . . . .	23
6.2.	Informative References . . . . .	23
6.3.	URL References . . . . .	24
	Authors' Addresses . . . . .	24

## 1. Introduction

With the rise of the Internet (especially the mobile Internet), Internet-oriented Web services become the mainstream of software architecture in the information age, but sophisticated web database construction and system management bring great increase of cost of software development, operation and maintenance. Whenever a system needs to be upgraded or updated, a large amount of coding workload is inevitable. With the rapid development of cloud computing and the growing popularity of SAAS, more and more users intend to migrate their software systems and deploy them to the cloud to get the cloud service resources they need. At the same time, there exists a common abstract model in the web information system, the system takes data management and maintenance as the core content, we can therefore extract its common features and provide a series of standards and requirements for the flexible construction and agile release of web software. In conclusion, cloud-oriented software migration and deployment technology has very important theoretical and practical significance. This draft focuses on the implementation standard of a cloud-oriented software migration and deployment technology, through which users can generate their own software system and migrate and

deploy to the cloud by simple operation. At the same time, it also provides an open RESTful API for users to obtain the required service resources through requests. From the development point of view, this kind of technology can transfer and deploy the software system conveniently and quickly, subtract the tedious and large number of coding work, and avoid repeated development. From the perspective of use, the convenient operation of this technology reduces the learning cost of users, has a strong usability, and has a unified front-end style. Users can access their own proprietary software system through the Internet, which is easy to promote. It also provides an expressive RESTful API that gives users the flexibility to adapt to complex and changing needs.

## 2. Definitions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The following definitions are for terms used in the context of this document.

- o "SAAS": SAAS is an abbreviation of Software-as-a-Service. It is a model for providing software over the Internet. The manufacturer deploys the application software on its own server, and customers can purchase the required application services according to their actual needs through the Internet, pay the cost to the manufacturers according to the quantity of service and the length of the order, and obtain the services provided by the manufacturers through the Internet.
- o "Cloud Computing": Cloud computing is a pay-per-use model that provides available, convenient, on-demand network access, the user enters a configurable computing resource sharing pool (resources include network, server, storage, application software, services), these resources can be provided quickly, with little administrative effort or little interaction with service providers.
- o "Cloud Services": Cloud services are models of the addition, use, and delivery of Internet-based related services, often involving the provision of dynamically scalable and often virtualized resources over the Internet.
- o "Cloud Storage": Cloud storage is a network computer data storage model. Data is stored on multiple virtual hosts and is generally held by a third party rather than on a dedicated server.

- o "Cloud Migration": Cloud migration refers to the migration of enterprises from traditional platforms to cloud platforms. Compared with traditional application platforms, cloud computing platforms have the advantages of powerful computing power, storage capacity, diverse services and high cost performance.
- o "Software Migration": This draft is aimed at migrating existing software systems that meet user requirements to a lightweight information management software migration cloud service platform. Systems migrated to the platform can be accessed over the Internet.
- o "Software Deployment": This draft is aimed at rapidly deploying a new software system that meets user requirements on the lightweight information management software migration cloud service platform. The software system can be accessed through the Internet.
- o "Content Management": Using advanced information technology to help users create, store, transmit, publish, apply and analyze content, and use computer technology to evolve and extract it to create more valuable content for enterprises and individuals.
- o "NoSQL": Do not use SQL database. Such databases refer to other types of databases other than traditional relational databases. This type of database is more consistent and can handle very large and high concurrent data. The techniques in this draft use NoSQL to provide storage.
- o "Distributed DataBase": It is a distributed file storage-based database designed to provide a scalable, high-performance data storage solution for web applications.
- o "Web API": Web Application Programming Interface. An application programming interface is an interface that allows users to access information from another service and integrate the service into its own application.
- o "REST": Representational State Transfer. A software architecture style, the service is abstracted into a set of discrete resources. Through the HTTP protocol, different request methods (POST, DELETE, PUT, GET) add, delete, modify, and query the resources specified by the URI.
- o "Restlet Framework": The Restlet project provides a lightweight and comprehensive framework for "building a mapping between REST concepts and Java classes". It can be used to implement any kind of RESTful system, not just RESTful web services.

- o "Multi-Tenant Technology": A software architecture technology, whose key goal is to share the same system or program components in a multi-user environment and to ensure logical isolation of data between users.
- o "Privilege Control": The permission control used in this draft refers to adopting the role-based access control principle and the group configuration permission template method to achieve the separation of the authority control from the business logic.
- o "Certification Security": Authentication security is part of the security of the system. The authentication security system used in this draft adopts the Cookie mechanism on the client side and the Session mechanism on the server side, and uses the MD5 message digest algorithm to achieve the authentication security of the entire session.
- o "Non-Programming Mode": Non-Programming Mode is defined as a kind of software development mode in this draft, referring to develop a software without any codes.

### 3. Technical Specific Implementation Standards

The main goal of this software migration deployment technology is to help users quickly deploy a specific software system of their own on a designated lightweight information management software development and migration cloud service platform, or to migrate the user's traditional software systems to the designated lightweight information management software migration cloud service platform. We propose an implementation standard for this technology so that users who need it can use it for software development, migration, and deployment.

Note: The cloud platforms described in the draft are all designated lightweight information management software development and migration cloud service platforms. The cloud platform administrator refers to the administrator of the cloud service platform for the designated lightweight information management software. The system administrator is the administrator of the software system on the cloud platform, also known as the tenant. The user is a normal user under each software system, and is lower than the system administrator level user.

#### 3.1. System Architecture Diagram

Figure 1 shows the working diagram of the framework.

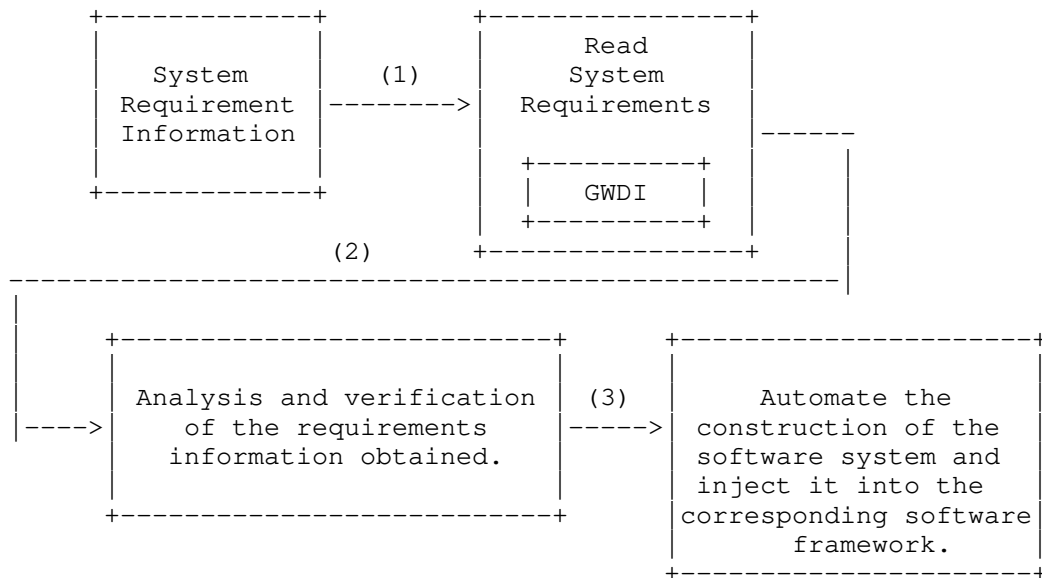


Figure 1:Reference Architecture

The standard requires a general Web Data Interface (GWDI). The system workflow is as follows:

Step (1) Requirements Submission: The user fills in the software system requirements information in the software system requirements table, and submits the request to GWDI through the browser.

Step (2) Demand Acquisition: GWDI obtains the software system requirements table submitted by the user, and reads the relevant information related to the requirements in the software system requirements table.

Step (3) Information Verification: GWDI analyzes and verifies the relevant information about the system requirements obtained.

Step (4) System Generation: Automatically construct a Web software system based on the verified requirements related information, and inject the system into the cloud platform.

### 3.2. SAAS Cloud Platform Related Implementation Standards

The technology MUST build a SAAS cloud platform that provides content management and data storage capabilities that can be rented to multiple tenants who are application developers. The platform includes the following modules:

- o Application Management Module
- o End User Management Module
- o Relationship Management Module between the End User and the Application
- o Resource Management Module
- o Fine-grained Access Rights Management Module

#### 3.2.1. Application Management Module

The application management module of the platform MUST support applications developed by management tenants, including:

- o Adding application
- o Deleting application

#### 3.2.2. End User Management Module

The end-user management module of the platform MUST support the management of end-user information for all applications of the tenant, including:

- o End-user registration
- o The end user changes the password
- o End-user logout

#### 3.2.3. Relationship Management Module between the End User and the Application

The relationship management module between the end user of the platform and the application MUST support the management of the use relationship between the end user and the application, including:

- o Establishment of user relationship

- o Elimination of the use relationship

#### 3.2.4. Resource Management Module

The resource management module of the platform MUST include at least:

- o Structured data resource management module

The structured data resource management module is used to manage the tenant's structured data in the cloud platform, including management of structured data table information, management of structured data table data resources. Each row of structured data resources will belong to a unique table, and will also belong to a unique user.

- o File resource management module

The file resource management module is used to manage the file resources of the tenant in the cloud platform. Each file resource will belong to the only application and the only user.

#### 3.2.5. Fine-grained Access Rights Management Module

The fine-grained access authority management module of the platform MUST support the management of end-user access rights to fine-grained resources within the tenant, including:

- o Structured form row data resource authority management module

The structured form row data resource authority management module includes:

- + For each structured table, configure the application of the structured table;
- + For each structured table, configure the access rights of the end user of the application to which the structured table belongs;
- + For each structured table, configure the access rights of the end user of the application that does not use the structured table;
- + For each structured table, configure the access rights of the end user to the single-row data resources that do not belong to the structured table.

- o Single file resource authority management module



The single file resource authority management module includes:

- + For each application, configure the access rights of the end user using the application to file resources within the application;
- + For each application, configure the access rights of the end user who does not use the application to the file resources within the application;
- + For each application, configure end-user access to files that are not part of the application.

### 3.2.6. Ways of Accessing Cloud Storage Resources

The platform SHOULD be able to access cloud storage resources in the following ways:

- o Through the POST, DELETE, PUT, GET requests initiated by REST API, the structured data resources specified by structured data URI are added, deleted, modified and queried.
- o Through the POST, DELETE and GET requests initiated by REST API, the file resources specified by file URI are uploaded, deleted and downloaded.

### 3.2.7. Format of Describing the Structured Data URI

The platform adopts the access method of the resources of the cloud storage platform described in 3.2.6, and it SHOULD be able to describe the structured data URI in the following format:

- o `/ {structure table name} [ / {filtered fields + combined form} / {conditions for filtered fields} ] ;`

Where, the structure table name is equivalent to the table name to be operated in the SQL statement. The contents in the brackets ([ ]) are optional: "NONE", which means all resources in the collection are manipulated. "YES", it represents an operation on a particular resource in the collection, equivalent to the WHERE clause in SQL.

### 3.2.8. Contents of the Ways of Accessing Cloud Storage Resources

The platform adopts a cloud storage platform resource access method described in 3.2.6, which SHOULD include:

- o Initiate PUT requests for structured data resources to modify the data resources specified by the URI, including: overwrite and

modify the specified fields, incremental changes to specified fields.

- o Initiate GET requests for structured data resources to query the data resources specified by the URI. Specific return formats can be set for the data resources that are queried, including: return by the specified field, return by the specified information page, and return by the specified field filtering.

### 3.3. Web Software Framework Related Implementation Standards

The standard MUST support a Web software framework that supports the software-as-a-service SAAS cloud pattern, where the software systems generated by the Web software framework are automatically deployed and distributed.

The Web software framework should specifically include the following views:

#### 3.3.1. System Administrator View

- o User Management

This view SHOULD be able to manage the user information of the Web software system, including the functions of adding users, removing users, adding user group information for users, and deleting user group information for users.

- o User Group Management

This view SHOULD be able to manage the user group information of the Web software system, including adding user groups, deleting user groups, adding user information for user groups, and deleting user information for user groups.

- o Model Management

This view SHOULD be able to manage the model information of the Web software system, including adding model information, deleting model information, adding model field information, deleting model field information, modifying model field information, and assigning different access rights to different user groups for the model.

### 3.3.2. User View

- o Data Management

This view SHOULD be able to manage the data table records of the Web software system, including adding data records, deleting data records, modifying data records, searching data records, and obtaining all data records.

- o Data Statistics

This view SHOULD be able to make statistics on the data table records of the Web software system, including functions such as record statistics, maximum statistics, minimum statistics, and average statistics for data table integral fields.

- o Import Excel and Other Format Tables

This view SHOULD be able to import data from Excel tables (sheets) and other format tables into the corresponding tables of the Web software system.

- o Export Excel and Other Formats

This view SHOULD be able to export data from the table of the Web software system to Excel tables (sheets) and tables in other formats.

- o Advanced Search

This view SHOULD be able to perform multiple logical combinatorial searches for multiple fields of the table to query the record information that satisfies the search criteria.

### 3.4. Implementation Standards on Construction Methods of Web Software

This standard SHALL support a table-driven software system automatic construction method within cloud mode, which is characterized by table submission, reading and verification, a Web software generation framework and a set of Web software generation workflow, in which the Web software generation framework should be the framework described in requirement 3.3.

### 3.4.1. System Requirements Table and Data Record Exchange Table

The table described in this method MUST include the system requirements table and the data record exchange table:

#### 3.4.1.1. System Requirements Table

The system requirements table MUST include the following parts:

- o System Administrator Information Section
  - \* System administrator ID
  - \* System administrator password
  - \* System title
- o User Group Information Section (multiple user groups can be customized in this table according to requirements)
  - \* User group ID
- o User Information Section (multiple users can be customized in this table according to requirements)
  - \* User ID
  - \* The user name
  - \* User password
  - \* The user belongs to the group (if filled in, it must be the item contained in the user group ID of the user group information section)
- o Model Information Section (multiple models can be customized in this table according to requirements)
  - \* Read and write permission of the user group to which it belongs
    - Permissions include: readable and writable, readable and unwritable, unreadable and unwritable
  - \* Other users' read and write rights
    - Permissions include: readable and writable, readable and unwritable, unreadable and unwritable

- \* Model name
- \* Subscriber ID (must be the item contained in the user ID in the user information section)
- \* Group ID (must be the item contained in the user group ID of the user group information section)
- \* Field type  
  
Field types include: Text, Float, Integer, Link, Date, Datetime, Boolean.
- \* Field name
- \* Whether empty is allowed  
  
Values include: TRUE and FALSE. Where TRUE means allowed to be empty;

#### 3.4.1.2. Data Record Exchange Table

The data record exchange table MUST include the following parts:

- o Field information section (all field names of the model to be exchanged for data records can be set according to requirements)
- o Data records section (all data records that need to be injected into the model can be added as required)

#### 3.4.2. Steps of Reading System Requirements Table

In the workflow of the Web software generation principle of this method, it MUST include the following steps:

1. Using the traversal unit of system requirements table to read through each cell of the system requirements table.
2. Using the reading unit of system administrator information to read system administrator ID, system administrator password, system title and other information when traversing system administrator information.
3. Using the reading unit of user group information to read user group ID information when traversing user group information.
4. Using the reading unit of user information to read user ID, user name, user password, user group ID (the user group ID must be

included in user group information) when traversing user information.

5. Using the reading unit of model information to read the information such as the reading and writing rights of the user group to which it belongs, the reading and writing rights of other users, the model name, the inputting ID, the group ID to which it belongs, the field type, the field name and whether it is allowed to be empty, when traversing the model information.

#### 3.4.3. Steps of Analyzing System Requirements Table

In the workflow of Web software generation of this method, it MUST include the analysis and verification flow of the relevant information about the system requirements acquired by the Web software framework, including the following parts:

1. Data type and format validation for information related to system requirements obtained.
2. As for the user information part of the system requirements table, if the user's group item is not empty, the framework should judge whether the filled value is included in the user group information part of the system requirements table.

#### 3.4.4. Contents of Modules for Users

The workflow of the Web software generation principle of this method MUST include the module for users to use in the cloud environment where the corresponding Web software framework is located, which must include the following contents:

1. The framework should use the API of dealing with system administrator information in the Read\_Demand class to read the system administrator section in the software system requirement table, each cell of which will be read successively, and then register the software system using the super administrator privileges with its ID and password and the system name, and further create relevant database entities of the software system in the underlying database of the Web software framework. At the same time, such four collections should be create in the database as user group, user, schema, and data.
2. The framework should use the API of dealing with user group information in the Read\_Demand class to read the user group information in the software system requirement table, each cell of which be read successfully, and generate json-formatted strings based on the group ID information obtained, and then

insert them into the user group set of database entities created in 1.

3. The framework should use the API of dealing with user information in the Read\_Demand class to read the user information in the software system requirement table, each cell of each row of which be read successfully, and generate json-formatted strings according to the user information obtained, and then insert them into the user set of database entities created in 1.
4. The framework should read the model information section of the system requirements table through the API used to process model information in the requirements table reading class (Read\_Demand class), read through the filled content in the model information cells, generate json-formatted strings based on model information, and insert them into the schema set of database entities created in 1.
5. The framework should operate the user group data in the group set of database entities in the underlying database through the user group management class (Manage\_Group) and return the results to the interface of the Web software framework. The Manage\_Group class implements the GManage\_Group interface defined in the Web software framework.
6. The framework should operate the user data in the user set of the database entity in the underlying database through the user management class (Manage\_User) and return the results to the interface of the Web software framework. The Manage\_User class implements the GManage\_User interface defined in the Web software framework.
7. The framework should operate the model data in the schema set of database entities in the underlying database through the model management class (Manage\_Schema) and return the results to the interface of the Web software framework. The Manage\_Schema class implements the GManage\_Schema interface defined in the Web software framework.
8. The framework should operate the data records in the data set of database entities in the underlying database through the data management class (Manage\_Data) and return the results to the interface of the Web software framework. The Manage\_Data class implements the GManage\_Data interface defined in the Web software framework.

#### 3.4.5. Steps of Principle of Constructing Web Software System

This method MUST satisfy the working principle of constructing Web software system automatically according to the system demand table. The principle includes the following steps:

1. The customer shall fill in the system requirements form described in 3.4.1.1 according to the requirements.
2. The Web software framework obtains the system requirements table uploaded by the customer in 1, and verifies whether the system requirements table conforms to the verification standard. If it does not conform to the verification standard, it prompts the user to have the wrong format, please re-upload it.
3. The Web software framework reads the system administrator information in the system requirements table.
4. Create a new system in the corresponding Web software framework according to the information read in 3, in which the administrator of the system is the system administrator in 3, and the system name is the system name in 3.
5. The Web software framework determines whether there is any part of user group information in the system requirements table. If not, it jumps to 8; if there is, it goes 6.
6. The Web software framework reads information about user groups in the system requirements table, including user group IDs.
7. According to the user group information read in 6, the user group is generated in the system created in 4, and the relevant information of the user group is the relevant information read in 6.
8. The Web software framework determines whether there is any relevant information about users in the system requirements table. If there is no information about users, it will jump to 11; if there is, it will go to 9.
9. The Web software framework reads relevant information of users in the system requirements table, including user ID, user name, user password, and user group.
10. The Web software framework generates users in the system created in 4 according to the user information read in 9, and the relevant information of the users is the information read in 9.



11. The Web software framework determines whether the relevant information of the model part exists in the system requirements table. If it does not, it will jump to 20, and if it does, it will go to 12.
12. The Web software framework reads the number of models in the system requirements table and assigns a current count of 0, then go to 13.
13. The Web software framework reads the relevant information of the model in the system requirements table, including the model name, model entry person, model group ID, group permissions, other user rights and other information.
14. According to the information related to the model read in 13, the model is generated in the system created in 4. The model name, model entry person, group ID of the model, group permission and other user rights are the information read in 13.
15. The Web software framework determines whether there is any information related to the structural fields in the model in the system requirements table. If there is no information, it will jump to 18; if there is, it will go to 16.
16. The Web software framework reads the relevant information of structural fields in the model in the system requirements table, including obtaining the name of structural fields of the model, field types, whether to allow null identification, default values and alternative values, etc.
17. The Web software framework creates model fields in the specified model according to the relevant information of structural fields in the model read in 16, and the field information is the information read in 16.
18. Current model number +1.
19. The Web software framework determines whether the current count is less than the number of models read in 12; if it is less than, it will jump back to 13; if it is not less than, it will go to 20.
20. The Web software framework generates the target Web software system.

#### 3.4.6. Steps of Injecting Data Records into Web Software System

The workflow of the Web software generation principle of this method MUST include the workflow of automatically injecting data records into the Web software system according to the data record exchange table, including the following steps:

1. The customer shall fill in the data record exchange form as required.
2. The Web software framework gets the data record exchange table uploaded by the customer, and verifies whether the data record exchange form meets the requirements of the data record exchange form. If it does not meet the requirements, it prompts the user to have the wrong format, please re-upload it.
3. The Web software framework reads the model field information in the data record exchange table.
4. The Web software framework determines the required data exchange model according to the model field information read in 3.
5. The Web software framework reads the data record information in the data record exchange table line by line and injects it into the model determined in 4 successively.

#### 3.4.7. Submission of System Requirements Table

The submission of the system requirements table SHALL include:

1. Online submission: according to the Web form format provided by the Web software generation framework, users fill in the system requirements information online and submit it to the Web software generation framework;
2. Offline submission: according to the offline form format template agreed by the Web software generation framework, users fill in the required software system demand information offline and submit it to the Web software generation framework by file upload.

#### 3.4.8. Submission of Data Exchange Table

The submission of the data exchange table SHALL include:

1. Online submission: after the user constructs a new Web software system according to the Web software generation framework, the user enters data into the Web page through the newly generated

Web software system, enters data information online and delivers it to the generated software system;

2. Offline submission: after the user constructs a new Web software system according to the Web software generation framework, the user enters the data record information offline and submits it to the generated software system through the data record exchange table template format provided by the newly generated Web software system by means of file upload.

### 3.5. Implementation Standards on Web Software Deployment

This framework MUST support the rapid deployment of a new set of software systems that meet the requirements on the cloud platform, where the deployment of software systems should support the deployment of the following modules:

#### 3.5.1. System Deployment

This framework SHOULD support two ways to deploy a system:

- o Supports registering users' own systems through cloud platform administrators to create system administrators belonging to users.
- o Supports users to register their own system through cloud platform and register their own system administrator.

#### 3.5.2. Model Deployment

This framework SHOULD enable system administrator users to enter their own software system and deploy the created models on the cloud platform, where the created models are configured as follows:

- o Model name, input personnel, and user group read and write permission are necessary when the model is created.
- o Field name, field type, whether the field is allowed to be empty, field default value, and field alternative value in the model are optional fields for creating the model.

#### 3.5.3. User Deployment

This framework SHOULD support to create new users affiliated to the software system created by a system administrator user on the cloud platform, where the creation of users is configured with user ID, user nickname, and user password.

#### 3.5.4. User Group Deployment

This framework SHOULD support to create new users affiliated to the software system created by a system administrator user on the cloud platform, where the creation of user groups is configured as follows:

- o The user group ID is necessary.
- o It is optional to select many users into the user group, but one user group must have one use at least.

#### 3.6. Implementation Standards on Web Software Migration

This framework SHOULD support the migration of existing software systems that meet the given requirements to the cloud platform, where the migration of software systems should support the migration of the following modules:

##### 3.6.1. Model Migration

This framework SHOULD support the migration of the specified model from the source system to the target software system, where the source container of the model includes at least one of the following:

1. Relational Database
  - \* SQLSERVER
  - \* MYSQL
  - \* ORACLE
2. Excel File. Here, the model migration may support two modes:
  - \* Migrating data from source to specify model structure
  - \* Migrating data from source to specify the model structure and migrating data within the model

##### 3.6.2. User Migration

This framework SHOULD support the migration of users in the source system and their access rights to the target software system, where the user's source container includes at least one of the following:

1. Relational Database
  - \* SQLSERVER

- \* MYSQL

- \* ORACLE

## 2. Excel File

### 3.6.3. User Group Migration

This framework SHOULD support the migration of user groups in the source system and their access rights to the target software system, where the user group's source container includes at least one of the following:

#### 1. Relational Database

- \* SQLSERVER

- \* MYSQL

- \* ORACLE

#### 2. Excel File

### 3.6.4. User Group Deployment

This framework SHOULD enable system administrator users to enter their own software systems and create user groups on the cloud platform, where the creation of user groups is configured as follows:

1. A user group ID created for identifying the user group is of necessity.
2. It is optional to select many users into a user group, but one user group must have one use at least.

## 4. Security Considerations

This draft proposes an implementation standard for software migration deployment technology for cloud environments, and does not make special requirements for the security of the technology itself. However, the security of the cloud platform and the security between different users in the software system are required. The security of the cloud platform is mainly authentication security, and can also be considered as session security to ensure the security of the user during using software. The security of different users in the system is called permission control. Data isolation between different systems, different user groups in the same system, and different resource access rights between different users should be considered.

## 5. IANA Considerations

This memo includes no request to IANA.

## 6. References

### 6.1. 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>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7705] George, W. and S. Amante, "Autonomous System Migration Mechanisms and Their Effects on the BGP AS\_PATH Attribute", RFC 7705, DOI 10.17487/RFC7705, November 2015, <<https://www.rfc-editor.org/info/rfc7705>>.
- [RFC8206] George, W. and S. Murphy, "BGPsec Considerations for Autonomous System (AS) Migration", RFC 8206, DOI 10.17487/RFC8206, September 2017, <<https://www.rfc-editor.org/info/rfc8206>>.

### 6.2. Informative References

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<https://www.rfc-editor.org/info/rfc2629>>.
- [RFC3347] Krueger, M. and R. Haagens, "Small Computer Systems Interface protocol over the Internet (iSCSI) Requirements and Design Considerations", RFC 3347, DOI 10.17487/RFC3347, July 2002, <<https://www.rfc-editor.org/info/rfc3347>>.
- [RFC6208] Sankar, K., Ed. and A. Jones, "Cloud Data Management Interface (CDMI) Media Types", RFC 6208, DOI 10.17487/RFC6208, April 2011, <<https://www.rfc-editor.org/info/rfc6208>>.
- [RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <<https://www.rfc-editor.org/info/rfc7322>>.

[RFC7491] King, D. and A. Farrel, "A PCE-Based Architecture for Application-Based Network Operations", RFC 7491, DOI 10.17487/RFC7491, March 2015, <<https://www.rfc-editor.org/info/rfc7491>>.

### 6.3. URL References

[idguidelines] IETF Internet Drafts editor, "<http://www.ietf.org/ietf/lid-guidelines.txt>".

[idnits] IETF Internet Drafts editor, "<http://www.ietf.org/ID-Checklist.html>".

[ietf] IETF Tools Team, "<http://tools.ietf.org>".

[ops] the IETF OPS Area, "<http://www.ops.ietf.org>".

[xml2rfc] XML2RFC tools and documentation, "<http://xml.resource.org>".

### Authors' Addresses

Can Yang (editor)  
South China University of Technology  
382 Zhonghuan Road East  
Guangzhou Higher Education Mega Centre  
Guangzhou, Panyu District  
P.R.China

Phone: +86 18602029601  
Email: [cscyang@scut.edu.cn](mailto:cscyang@scut.edu.cn)

Shiying Pan (editor)  
South China University of Technology  
382 Zhonghuan Road East  
Guangzhou Higher Education Mega Centre  
Guangzhou, Panyu District  
P.R.China

Email: [201721041376@scut.edu.cn](mailto:201721041376@scut.edu.cn)

Haibo Sun  
Inspur  
163 Pingyun Road  
Guangzhou, Tianhe District  
P.R.China

Email: sunhb@inspur.com

Kemin Qu  
NetEase, Inc  
Netease Building, Building E, Guangzhou Information Port  
16 Keyun Road  
Guangzhou, Tianhe District  
P.R.China

Email: gzqukemin@corp.netease.com

Guoqiang Han  
South China University of Technology  
382 Zhonghuan Road East  
Guangzhou Higher Education Mega Centre  
Guangzhou, Panyu District  
P.R.China

Email: csgqhan@scut.edu.cn



CORE WG  
Internet-Draft  
Intended status: Informational  
Expires: May 11, 2019

Z. Cao  
K. Jin  
B. Fu  
D. Zhang  
Huawei  
November 7, 2018

Speeding Up CoAP Block-wise Transfer  
draft-zcao-core-speedy-blocktran-00

Abstract

This document specifies a method to speed up block-wise transfer in CoAP. With this, the client can indicate its willingness to be responded with a sequence of blocks without issuing request for each block one by one.

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 11, 2019.

Copyright Notice

Copyright (c) 2018 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. Conventions used in this document . . . . .	2
3. Speedy Blockwise Transfer . . . . .	2
3.1. The Speedy Block Option . . . . .	2
3.2. Example Follow-chart . . . . .	4
3.3. Retransmission Considerations . . . . .	5
4. Security Considerations . . . . .	6
5. IANA Considerations . . . . .	6
6. Acknowledgments . . . . .	7
7. References . . . . .	7
Authors' Addresses . . . . .	7

## 1. Introduction

When performing the block-wise CoAP transfer as defined in [RFC7959], the Client needs to continuously send requests to the Server, using the BLOCK options to specify the exact segment that is expected. As a consequence, the Server can only acknowledge different blocks one by one according to the request. Such a design was a reasonable choice since the server can be implemented to be truly stateless and lightweight. However, there are some cases where the server is capable of keeping necessary states so that the transfer can be performed in a more efficiently and faster way, e.g., the server being a firmware upgrade device without constraints defined for CoAP. This document describes such a proposal that is used to speed up the CoAP block transfer.

## 2. Conventions used in this document

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

## 3. Speedy Blockwise Transfer

### 3.1. The Speedy Block Option

We introduce a new option called 'BlockS', the format being specified in Figure 1. The client can include this option in the payload of the request to indicate its willingness to fetch the block content in a speedy way.

No.	C	U	N	R	Name	Format	Length	Default
TBD	C	U	-	-	BlockS	uint	0-4	(none)

Figure 1: The Speedy Block Option

The value of the BlockS option is a variable-size unsigned integer. This integer value encodes these four fields: NUM, M, SIZE, and SPDYWND; NUM, M and SIZE are defined the same as [RFC7959].

NUM: the relative number of the block (NUM) within a sequence of blocks with the given size. (4, 12, 20 bit)

M: whether more blocks are following (M) (1-bit);

SIZE: the size of the block (SIZX) (3-bit), the actually size in octet equals to  $(2^{SIZX} - 1)$ ;

SPDYWND(8-bit): the allowance of maximum window size used for speed-up, for a total of SPDYWND messages, the Server will solicitate an acknowledgement from the client

if the client includes the BlockS option in its request, it indicates it will be willing and capable to receive a bunch of blocks without sending the request one by one. Since the client is fully aware of its capacity, it MUST indicate the SPDYWND value in the BlockS option explicitly.

Upon receiving a request with the BlockS option, the Server will check the SPDYWND value in the request. It will calculate how many blocks (according to the SIZE value) the requested content consists of, and if this value being smaller than the SPDYWND, it will send the first block by changing the SPDYWND to the actually size (in block).

The first block is piggyback in the Acknowledgement of the client request. for the next (SPDYWND-1) messages, the server will send the content in a NON message. for every SPDYWND responding messages, the server will send a CON message to solicitate an acknowledgement from the client to make sure that the client is still alive.

if the Server does not support the BlockS option, it will neglect this message. The client, under this case, needs to implement a timeout mechanism, so that it can switch to other block-wise transfer (Block1 or Block2) as specified in [RFC7959].

## 3.2. Example Follow-chart

In the following example, the client specifies its SPDYWND as 5, but the server finds out the representation only consists of 4 blocks, it will change the SPDYWND value to 4 in the BlockS option. The client will recognize this as an indication of actually length of the resource, continue to receive the content response until the M(More) bit being unset.

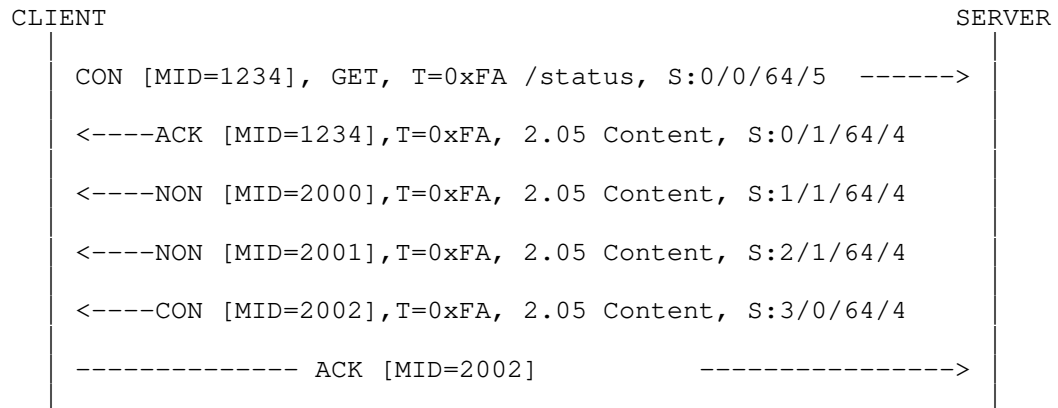


Figure 2: Speedy Blockwise GET with Early Negotiation

In the following example, the client specifies its SPDYWND as 5, and the server finds out the representation consists of 8 blocks, it will keep the SPDYWND value to 5 in the BlockS option. The server will send a CON message every SPDYWND messages.

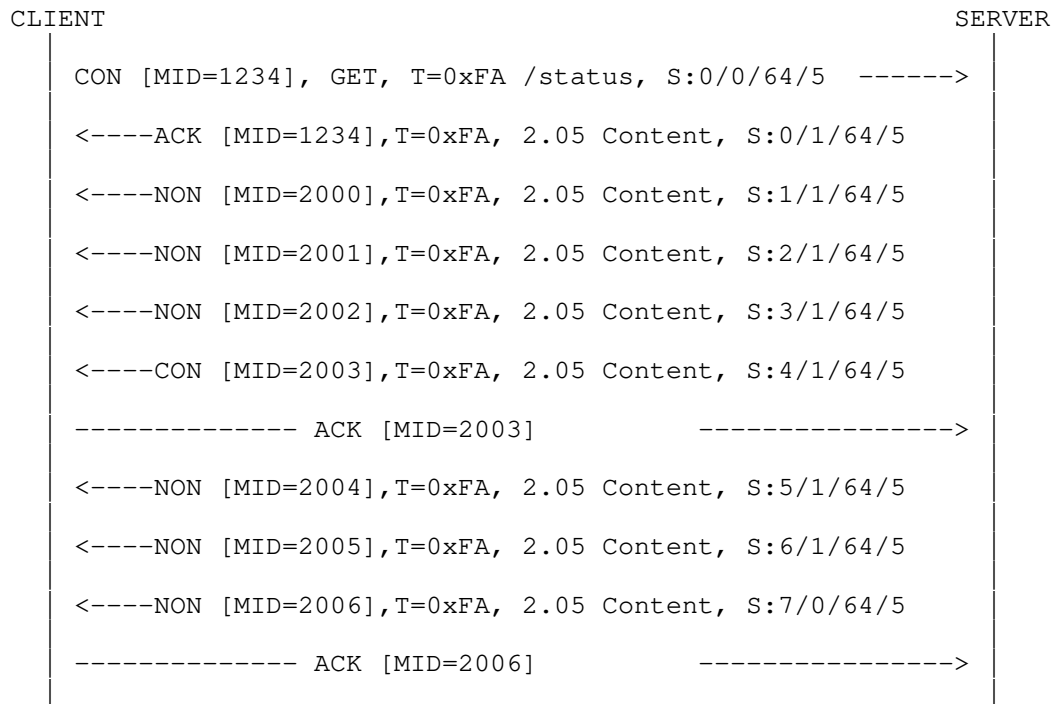


Figure 3: Speedy Blockwise GET with Early Negotiation

### 3.3. Retransmission Considerations

There is only one CON message every SPDYWND messages sending by the server. if the NON messages get lost during the communication, the server has no way to know the fact and will not retransmit it. Only after receiving a number of messages from the server, the Client can identify the "hole" due to packet loss. The client can send separate request per the missing sequence number in the whole block.

Figure 4 shows an example where the NUM 2 and NUM 6 blocks get lost. The client will send separate requests for block number 2 and 6 after receiving the final block where M=0.

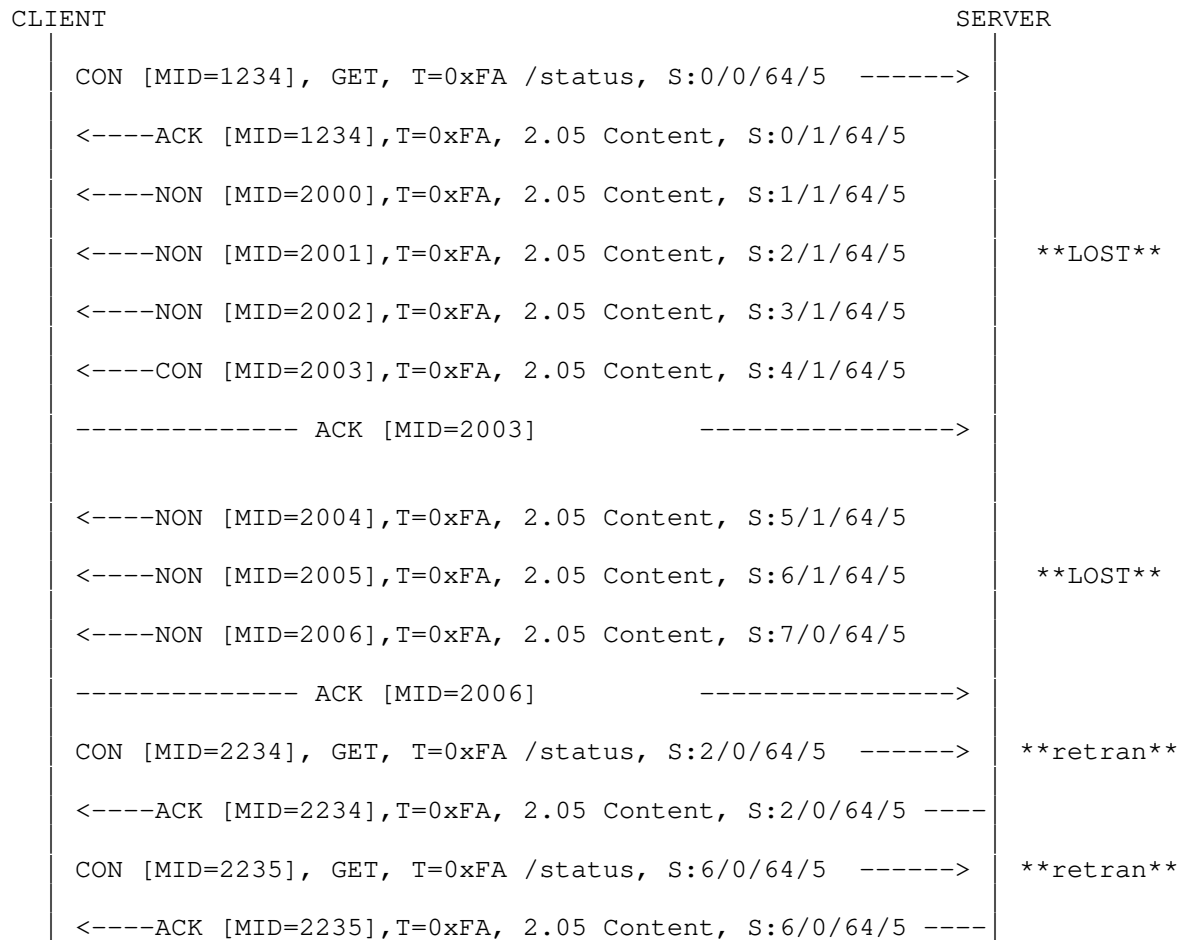


Figure 4: Retransmission Flow

#### 4. Security Considerations

TBD

#### 5. IANA Considerations

This document needs an assignment of the BlockS option value defined in Section.3.1.

## 6. Acknowledgments

TBD

## 7. 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>>.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [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>>.

## Authors' Addresses

Zhen Cao  
Huawei  
Xinxi Rd 3  
Beijing 100085  
China

Email: [zhencao.ietf@gmail.com](mailto:zhencao.ietf@gmail.com)

Ke Jin  
Huawei  
Shenzhen  
China

Email: [jinke@huawei.com](mailto:jinke@huawei.com)

Baicheng Fu  
Huawei  
Shenzhen  
China

Email: [290275570@qq.com](mailto:290275570@qq.com)

Dacheng Zhang  
Huawei  
Beijing  
China

Email: zhangdacheng@huawei.com