

CoRE Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: January 1, 2019

C. Bormann
Universitaet Bremen TZI
K. Hartke
Ericsson
June 30, 2018

Proactively Assigning CoAP Content-Format Numbers to Registered Media
Types
draft-bormann-core-proactive-ct-00

Abstract

In order to use a media type with the Constrained Application Protocol (CoAP), a numeric identifier needs to be registered for it, the Content-Format number.

RFC 7252 defines registration procedures for Content-Format numbers. The present document defines a proactive procedure to register a Content-Format number for many of the media types that are registered and discusses the benefits and limitations of that approach.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 1, 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	3
2. Media Types and Content-Format Numbers	3
3. Procedure	3
4. Discussion	4
4.1. Latency	4
4.2. Potential Mishaps	4
4.2.1. Race Conditions	4
4.2.2. Depletion of Pre-Registration Space	5
5. Instructions to the Designated Expert	6
6. IANA Considerations	6
7. Security Considerations	6
Acknowledgements	6
9. References	7
9.1. Normative References	7
9.2. Informative References	7
Authors' Addresses	7

1. Introduction

To identify representation formats in a concise form, the Constrained Application Protocol (CoAP) uses numeric identifiers, the Content-Format Numbers [RFC7252]. A Content-Format number identifies a media type [RFC6838] and a content coding (usually the identity coding). Content-Format numbers are assigned in the CoAP Content-Formats Registry, as defined in Section 12.3 of [RFC7252].

At the time of writing, a couple dozen Content-Format numbers are registered. Any new application that needs to register new media types for use with CoAP can define the Content-Format numbers for its media types as well. However, using existing applications and their media types over CoAP is complicated by the need to register the Content-Format numbers for these media types.

As of 2018, less than 2000 media types have been registered in the Media Type Registry managed by [RFC6838], in a registry established by [RFC1590] in 1994. No trends significantly accelerating the growth of this registry are currently anticipated.

The size of the space available for Content-Format numbers is a 16-bit unsigned number. It is therefore very well possible to go ahead and pre-define a Content-Format number for each media type (where possible). When CoAP was defined, the space between 1000 and 9999 was informally set aside for this purpose (as part of the space between 256 and 9999 that is reserved for future use in IETF specifications, with IETF Review or IESG Approval).

The present document defines how the assignment of Content-Format numbers for each existing media type and media type registered in the future is performed.

1.1. Terminology

This memo uses terms from [RFC7252] and [RFC6838].

2. Media Types and Content-Format Numbers

A Content-Format number identifies a media type with all the media type parameters, as well as a content-coding to be used with the media type. E.g., Content-Format 0 stands for "text/plain; charset=utf-8" with identity content-coding.

It is generally not easy to extract from its registration the parameters and ranges of parameter values that will be used with a media type. The present document therefore only attempts to handle media type parameters for one specific case: Where a charset needs to be defined, this is always set to "utf-8".

Where parameters other than charset are needed by an application, it will continue to need to register Content-Format numbers despite the proactive registration defined here.

Similarly, any content-coding beyond identity will need to be defined in a separate registration.

Therefore, the proactive registration procedure defined in this document defines a single Content-Format number for each media type, with no parameters (or just charset), and with identity content-coding. This number is assigned by the below Procedure to fall in the space between 1000 and 9999.

3. Procedure

Content-Format numbers need to be assigned for each existing and new media type. Instead of defining a detailed procedure for this, the present document delegates the definition of the procedure to a designated expert.

The designated expert publishes the list of proactively registered Content-Format numbers regularly at <https://svn.tools.ietf.org/svn/wg/core/mediatypes.txt>

The designated expert is requested to

- o only ever add information to the proactive registration document (no changes or deletions)
- o ensure that the proactive registration document is updated in some reasonable cadence (e.g., monthly)
- o provide a way to effect a quick update if such an update is reasonably called for
- o alert IANA to such updates.

IANA regularly (and when alerted) pulls the published list of registrations, detects additions, and adds those additions to its Content-Format registry.

4. Discussion

4.1. Latency

New media types do not immediately cause an update of the pre-registration list, and such an update does not immediately case new Content-Format number registries. Where this latency becomes an issue (e.g., because of deadlines of other standards development organizations that depend on these procedures), the designated expert can be alerted to effect a quick update.

New media types that are expressly intended for use in constrained environments of course should not wait for the procedure described here to effect their content-format registration, but should include the registration of a Content-Format number in their IANA considerations, as before. This also makes sure that any additional considerations (such as the potential need for a single-byte content-format number) are taken into account.

4.2. Potential Mishaps

4.2.1. Race Conditions

When the IANA registers a new media type and associated content-format numbers, the registry state could briefly show the new media type but not the new content-format numbers. If an update is created

to the pre-registrations at this very moment, the assignment of redundant Content-Format numbers could not be prevented.

The present document does not attempt to prevent the registration of redundant Content-Format numbers. So, "application/json" is both identified by Content-Format number 50 [RFC7252] and by the Content-Format number 4330 assigned under the pre-registration procedure.

4.2.2. Depletion of Pre-Registration Space

When a survey was run June 2018, 1726 media types were registered. 1006 of these have no parameters and will be proactively assigned a Content-Format number under this scheme. 276 more have just one parameter, "charset", and will also be proactively assigned a Content-Format number by setting this parameter to "utf-8".

418 media types have parameters that would require manual assignment of appropriate parameter values. This is not envisioned for the scheme described in this document. Finally, 26 media types could not easily be automatically analyzed and would require manual processing before sorting into one of the categories; this document leaves it up to the designated expert to decide whether to perform this processing and where.

In summary, as of the time of the survey, about 1/7 of the space envisioned for the scheme will be used by the media type registrations performed in the first 24 years of the registry (and the entire space reserved in turn is a bit less than 1/7 of the total space for Content-Format numbers).

Sudden changes in the patterns of media type registrations, although not anticipated at this time, could lead to depletion of the pre-registration space. This would not be a disaster, but would simply return Content-Format number registration to the situation before proactive registration (with the existing assignment of course continuing to be usable). The present document does not attempt to define a solution for this unlikely case.

However, the pre-registration procedure could motivate a malicious actor to define a large number of media types just to cause this depletion. One would hope that this is already prevented by the media type registration procedures, but just to reduce the incentive for such an attack, the procedures defined in this document make use of a designated expert that could detect such an attack and allow the designated expert to apply some mitigation.

5. Instructions to the Designated Expert

The designated expert is instructed to operate along the lines of the procedure described in Section 3, and towards the objectives defined in this document.

Between these two, the objectives are the overriding concern. Where the procedure turns out to no longer serve to further the objectives, the designated expert is instructed to adapt the procedure. If substantive changes to the procedure are deemed necessary, the designated expert is instructed to raise a discussion on the mailing list "core-parameters@ietf.org"; if the result of the discussion is a change of moderate extent, the designated expert can simply perform that change, document it on the mailing list, and act based on the updated procedure.

(If several designated experts are appointed, the above requires consensus between the designated experts.)

Fundamental changes, e.g., stepping out of the boundary of the number space, require further IETF review.

6. IANA Considerations

This entire document is about a IANA procedure.

7. Security Considerations

Accurate identification of representation formats can be important for security. Lowering the threshold for obtaining the registrations needed for this identification can therefore have a positive security impact. Conversely, limiting the representation formats pre-registered for each media type to just the single case without parameters and with identity content-coding might encourage imprecise identification. The present document is therefore not to be used as a substitute for registering any more specific Content-Format numbers needed by an application.

Procedures as defined in the present document can also be the subject of attacks. See Section 4.2.2 for one such consideration.

Acknowledgements

TBD

9. References

9.1. Normative References

- [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>>.
- [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>>.

9.2. Informative References

- [RFC1590] Postel, J., "Media Type Registration Procedure", RFC 1590, DOI 10.17487/RFC1590, March 1994, <<https://www.rfc-editor.org/info/rfc1590>>.

Authors' Addresses

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

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

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

6TiSCH Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 26, 2018

M. Vucinic, Ed.
University of Montenegro
J. Simon
Analog Devices
K. Pister
University of California Berkeley
M. Richardson
Sandelman Software Works
May 25, 2018

Minimal Security Framework for 6TiSCH
draft-ietf-6tisch-minimal-security-06

Abstract

This document describes the minimal framework required for a new device, called "pledge", to securely join a 6TiSCH (IPv6 over the TSCH mode of IEEE 802.15.4e) network. The framework requires that the pledge and the JRC (join registrar/coordinator, a central entity), share a symmetric key. How this key is provisioned is out of scope of this document. Through a single CoAP (Constrained Application Protocol) request-response exchange secured by OSCORE (Object Security for Constrained RESTful Environments), the pledge requests admission into the network and the JRC configures it with link-layer keying material and other parameters. The JRC may at any time update the parameters through another request-response exchange secured by OSCORE. This specification defines the Constrained Join Protocol and its CBOR (Concise Binary Object Representation) data structures, a new Stateless-Proxy CoAP option, and configures the rest of the 6TiSCH communication stack for this join process to occur in a secure manner. Additional security mechanisms may be added on top of this minimal framework.

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 November 26, 2018.

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. Terminology	4
3. Identifiers	5
4. One-Touch Assumption	5
5. Join Process Overview	7
5.1. Step 1 - Enhanced Beacon	8
5.2. Step 2 - Neighbor Discovery	9
5.3. Step 3 - Constrained Join Protocol (CoJP) Execution	9
5.4. The Special Case of the 6LBR Pledge Joining	10
6. Link-layer Configuration	10
7. Network-layer Configuration	10
7.1. Identification of Join Request Traffic	11
7.2. Identification of Join Response Traffic	12
8. Application-level Configuration	12
8.1. OSCORE Security Context	13
9. Constrained Join Protocol (CoJP)	15
9.1. Join Exchange	16
9.2. Parameter Update Exchange	18
9.3. CoJP Objects	19
9.4. Parameters	27
9.5. Mandatory to Implement Algorithms	28
10. Stateless-Proxy CoAP Option	28
11. Security Considerations	29
12. Privacy Considerations	30
13. IANA Considerations	30
13.1. CoAP Option Numbers Registry	31
13.2. CoJP Parameters Registry	31
13.3. CoJP Key Usage Registry	31
14. Acknowledgments	32

15. References	33
15.1. Normative References	33
15.2. Informative References	33
Appendix A. Example	35
Authors' Addresses	37

1. Introduction

This document presumes a 6TiSCH network as described by [RFC7554] and [RFC8180]. By design, nodes in a 6TiSCH network [RFC7554] have their radio turned off most of the time, to conserve energy. As a consequence, the link used by a new device for joining the network has limited bandwidth [RFC8180]. The secure join solution defined in this document therefore keeps the number of over-the-air exchanges for join purposes to a minimum.

The micro-controllers at the heart of 6TiSCH nodes have a small amount of code memory. It is therefore paramount to reuse existing protocols available as part of the 6TiSCH stack. At the application layer, the 6TiSCH stack already relies on CoAP [RFC7252] for web transfer, and on OSCORE [I-D.ietf-core-object-security] for its end-to-end security. The secure join solution defined in this document therefore reuses those two protocols as its building blocks.

This document defines a secure join solution for a new device, called "pledge", to securely join a 6TiSCH network. The specification defines the Constrained Join Protocol (CoJP) used by the pledge to request admission into a network managed by the JRC, and for the JRC to configure the pledge with the necessary parameters and update them at a later time, a new CoAP option, and configures different layers of the 6TiSCH protocol stack for the join process to occur in a secure manner.

The Constrained Join Protocol defined in this document is generic and can be used as-is in modes of IEEE Std 802.15.4 other than TSCH, that 6TiSCH is based on. The Constrained Join Protocol may as well be used in other (low-power) networking technologies where efficiency in terms of communication overhead and code footprint is important. In such a case, it may be necessary to register configuration parameters specific to the technology in question, through the IANA process. The overall join process described in Section 5 and the configuration of the stack is, however, specific to 6TiSCH.

The Constrained Join Protocol assumes the presence of a JRC (join registrar/coordinator), a central entity. It further assumes that the pledge and the JRC share a symmetric key, called PSK (pre-shared key). The PSK is used to configure OSCORE to provide a secure channel to CoJP. How the PSK is installed is out of scope of this

document: this may happen through the one-touch provisioning process or by a key exchange protocol that may precede the execution of the 6TiSCH Join protocol.

When the pledge seeks admission to a 6TiSCH network, it first synchronizes to it, by initiating the passive scan defined in [IEEE802.15.4]. The pledge then exchanges messages with the JRC; these messages can be forwarded by nodes already part of the 6TiSCH network. The messages exchanged allow the JRC and the pledge to mutually authenticate, based on the PSK. They also allow the JRC to configure the pledge with link-layer keying material, link-layer short address and other parameters. After this secure join process successfully completes, the joined node can interact with its neighbors to request additional bandwidth using the 6top Protocol [I-D.ietf-6tisch-6top-protocol] and start sending the application traffic.

2. 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]. These words may also appear in this document in lowercase, absent their normative meanings.

The reader is expected to be familiar with the terms and concepts defined in [I-D.ietf-6tisch-terminology], [RFC7252], [I-D.ietf-core-object-security], and [RFC8152].

The specification also includes a set of informative specifications using the Concise data definition language (CDDL) [I-D.ietf-cbor-cddl].

The following terms defined in [I-D.ietf-6tisch-terminology] are used extensively throughout this document:

- o pledge
- o joined node
- o join proxy (JP)
- o join registrar/coordinator (JRC)
- o enhanced beacon (EB)
- o join protocol

- o join process

The following terms defined in [RFC6775] are also used throughout this document:

- o 6LoWPAN Border Router (6LBR)

The term "6LBR" is used interchangeably with the term "DODAG root" defined in [RFC6550], assuming the two entities are co-located, as recommended by [I-D.ietf-6tisch-architecture].

The term "pledge", as used throughout the document, explicitly denotes non-6LBR devices attempting to join over an IEEE Std 802.15.4 network interface. The device that attempts to join as the 6LBR of the network and does so over another network interface is explicitly denoted as the "6LBR pledge". When the text equally applies to the pledge and the 6LBR pledge, the "(6LBR) pledge" form is used.

In addition, we use the generic terms "network identifier" and "pledge identifier". See Section 3.

3. Identifiers

The "network identifier" uniquely identifies the 6TiSCH network in the namespace managed by a JRC. Typically, this is the 16-bit Personal Area Network Identifier (PAN ID) defined in [IEEE802.15.4]. Companion documents can specify the use of a different network identifier for join purposes, but this is out of scope of this specification. Such identifier needs to be carried within Enhanced Beacon (EB) frames.

The "pledge identifier" uniquely identifies the (6LBR) pledge in the namespace managed by a JRC. The pledge identifier is typically the globally unique 64-bit Extended Unique Identifier (EUI-64) of the IEEE Std 802.15.4 device. This identifier is used to generate the IPv6 addresses of the (6LBR) pledge and to identify it during the execution of the join protocol. For privacy reasons, it is possible to use an identifier different from the EUI-64 (e.g. a random string). See Section 12.

4. One-Touch Assumption

This document assumes a one-touch scenario. The (6LBR) pledge is provisioned with certain parameters before attempting to join the network, and the same parameters are provisioned to the JRC.

There are many ways by which this provisioning can be done. Physically, the parameters can be written into the (6LBR) pledge

using a number of mechanisms, such as a JTAG interface, a serial (craft) console interface, pushing buttons simultaneously on different devices, over-the-air configuration in a Faraday cage, etc. The provisioning can be done by the vendor, the manufacturer, the integrator, etc.

Details of how this provisioning is done is out of scope of this document. What is assumed is that there can be a secure, private conversation between the JRC and the (6LBR) pledge, and that the two devices can exchange the parameters.

Parameters that are provisioned to the (6LBR) pledge include:

- o Pre-Shared Key (PSK). The JRC additionally needs to store the pledge identifier bound to the given PSK. The PSK SHOULD be at least 128 bits in length, generated uniformly at random. It is RECOMMENDED to generate the PSK with a cryptographically secure pseudorandom number generator. Each (6LBR) pledge SHOULD be provisioned with a unique PSK.
- o Optionally, a network identifier. Provisioning the network identifier is RECOMMENDED. However, due to the operational constraints the network identifier may not be known at the time when the provisioning is done. In case this parameter is not provisioned to the pledge, the pledge attempts to join one network at a time, which significantly prolongs the join process. In case this parameter is not provisioned to the 6LBR pledge, the 6LBR pledge can receive it from the JRC as part of the join protocol.
- o Optionally, any non-default algorithms. The default algorithms are specified in Section 9.5. When algorithm identifiers are not exchanged, the use of these default algorithms is implied.

Additionally, the 6LBR pledge that is not co-located with the JRC needs to be provisioned with:

- o Global IPv6 address of the JRC. This address is used by the 6LBR pledge to address the JRC during the join process. The 6LBR pledge may also obtain the IPv6 address of the JRC through other available mechanisms, such as DHCPv6, GRASP, mDNS, the use of which is out of scope of this document. Pledges do not need to be provisioned with this address as they discover it dynamically during the join process.

5. Join Process Overview

This section describes the steps taken by a pledge in a 6TiSCH network. When a pledge seeks admission to a 6TiSCH network, the following exchange occurs:

1. The pledge listens for an Enhanced Beacon (EB) frame [IEEE802.15.4]. This frame provides network synchronization information, and tells the device when it can send a frame to the node sending the beacons, which plays the role of Join Proxy (JP) for the pledge, and when it can expect to receive a frame. The Enhanced Beacon provides the L2 address of the JP and it may also provide its link-local IPv6 address.
2. The pledge configures its link-local IPv6 address and advertises it to the JP using Neighbor Discovery. This step may be omitted if the link-local address has been derived from a known unique interface identifier, such as an EUI-64 address.
3. The pledge sends a Join Request to the JP in order to securely identify itself to the network. The Join Request is forwarded to the JRC.
4. In case of successful processing of the request, the pledge receives a Join Response from the JRC (via the JP). The Join Response contains configuration parameters necessary for the pledge to join the network.

From the pledge's perspective, joining is a local phenomenon - the pledge only interacts with the JP, and it needs not know how far it is from the 6LBR, or how to route to the JRC. Only after establishing one or more link-layer keys does it need to know about the particulars of a 6TiSCH network.

The join process is shown as a transaction diagram in Figure 1:

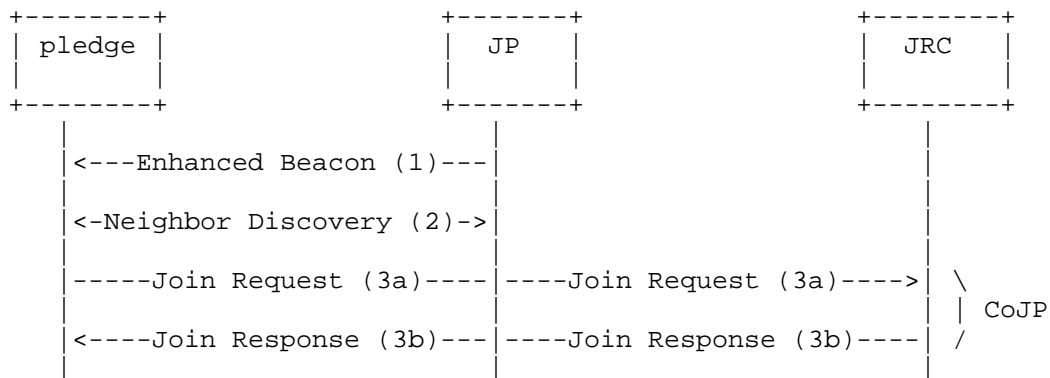


Figure 1: Overview of a successful join process. CoJP stands for Constrained Join Protocol.

As other nodes in the network, the 6LBR node plays the role of the JP. The 6LBR may in addition be co-located with the JRC.

The details of each step are described in the following sections.

5.1. Step 1 - Enhanced Beacon

The pledge synchronizes to the network by listening for, and receiving, an Enhanced Beacon (EB) sent by a node already in the network. This process is entirely defined by [IEEE802.15.4], and described in [RFC7554].

Once the pledge hears an EB, it synchronizes to the joining schedule using the cells contained in the EB. The pledge can hear multiple EBs; the selection of which EB to use is out of the scope for this document, and is discussed in [RFC7554]. Implementers should make use of information such as: what network identifier the EB contains, whether the source link-layer address of the EB has been tried before, what signal strength the different EBs were received at, etc. In addition, the pledge may be pre-configured to search for EBs with a specific network identifier.

If the pledge is not provisioned with the network identifier, it attempts to join one network at a time, as described in Section 9.1.3.

Once the pledge selects the EB, it synchronizes to it and transitions into a low-power mode. It follows the provided schedule which indicates the slots that the pledge may use for the join process. During the remainder of the join process, the node that has sent the EB to the pledge plays the role of JP.

At this point, the pledge may proceed to step 2, or continue to listen for additional EBs.

5.2. Step 2 - Neighbor Discovery

The pledge forms its link-local IPv6 address based on the interface identifier, as per [RFC4944]. The pledge MAY perform the Neighbor Solicitation / Neighbor Advertisement exchange with the JP, as per Section 5.5.1 of [RFC6775]. The pledge and the JP use their link-local IPv6 addresses for all subsequent communication during the join process.

Note that Neighbor Discovery exchanges at this point are not protected with link-layer security as the pledge is not in possession of the keys. How JP accepts these unprotected frames is discussed in Section 6.

5.3. Step 3 - Constrained Join Protocol (CoJP) Execution

The pledge triggers the join exchange of the Constrained Join Protocol (CoJP). The join exchange consists of two messages: the Join Request message (Step 3a), and the Join Response message conditioned on the successful security processing of the request (Step 3b). All CoJP messages are exchanged over a secure channel that provides confidentiality, data authenticity and replay protection.

5.3.1. Step 3a - Join Request

The Join Request is a message sent from the pledge to the JP, and which the JP forwards to the JRC. The pledge indicates in the Join Request the role it requests to play in the network as well as the identifier of the network it requests to join. The JP forwards the Join Request to the JRC on the existing 6TiSCH network. How exactly this happens is out of scope of this document; some networks may wish to dedicate specific slots for this join traffic.

5.3.2. Step 3b - Join Response

The Join Response is sent by the JRC to the pledge, and is forwarded through the JP. The packet containing the Join Response travels from the JRC to JP using the operating routes in the 6TiSCH network. The JP delivers it to the pledge. The JP operates as the application-layer proxy, and does not keep any state to forward the message.

The Join Response contains different parameters needed by the pledge to become a fully operational network node. For example, these parameters are the link-layer key(s) currently in use in the network,

the short link-layer address assigned to the pledge, the IPv6 address of the JRC needed by the pledge to operate as the JP, and others.

5.4. The Special Case of the 6LBR Pledge Joining

The 6LBR pledge performs Section 5.3 of the join process described above, just as any other pledge, albeit over another network interface. There is no JP intermediating the communication between the 6LBR pledge and the JRC, as described in Section 7. The other steps of the described join process do not apply to the 6LBR pledge. How the 6LBR pledge obtains an IPv6 address and triggers the execution of the CoJP protocol is out of scope of this document.

6. Link-layer Configuration

In an operational 6TiSCH network, all frames MUST use link-layer frame security [RFC8180]. The IEEE Std 802.15.4 security attributes MUST include frame authenticity, and MAY include frame confidentiality (i.e. encryption).

The pledge does not initially do any authenticity check of the EB frames, as it does not possess the link-layer key(s) in use. The pledge is still able to parse the contents of the received EBs and synchronize to the network, as EBs are not encrypted [RFC8180].

When sending frames during the join process, the pledge sends unencrypted and unauthenticated frames. The JP accepts these unsecured frames for the duration of the join process. This behavior may be implemented by setting the "secExempt" attribute in the IEEE Std 802.15.4 security configuration tables. How the JP learns whether the join process is ongoing is out of scope of this specification.

As the EB itself cannot be authenticated by the pledge, an attacker may craft a frame that appears to be a valid EB, since the pledge can neither verify the freshness nor verify the address of the JP. This opens up a possibility of DoS attack, as discussed in Section 11.

7. Network-layer Configuration

The pledge and the JP SHOULD keep a separate neighbor cache for untrusted entries and use it to store each other's information during the join process. Mixing neighbor entries belonging to pledges and nodes that are part of the network opens up the JP to a DoS attack, as the attacker may fill JP's neighbor table and prevent the discovery of legitimate neighbors. How the pledge and the JP decide to transition each other from untrusted to trusted cache, once the join process completes, is out of scope. One implementation

technique is to use the information whether the incoming frames are secured at the link layer.

The pledge does not communicate with the JRC at the network layer. This allows the pledge to join without knowing the IPv6 address of the JRC. Instead, the pledge communicates with the JP at the network layer using link-local addressing, and with the JRC at the application layer, as specified in Section 8.

The JP communicates with the JRC over global IPv6 addresses. The JP discovers the network IPv6 prefix and configures its global IPv6 address upon successful completion of the join process and the obtention of link-layer keys. The pledge learns the actual IPv6 address of the JRC from the Join Response, as specified in Section 9.1.2; it uses it once joined in order to operate as a JP.

As a special case, the 6LBR pledge is expected to have an additional network interface that it uses in order to obtain the configuration parameters from the JRC and start advertising the 6TiSCH network. This additional interface needs to be configured with a global IPv6 address, by a mechanism that is out of scope of this document. The 6LBR pledge uses this interface to directly communicate with the JRC using global IPv6 addressing.

The JRC can be co-located on the 6LBR. In this special case, the IPv6 address of the JRC can be omitted from the Join Response message for space optimization. The 6LBR then MUST set the DODAGID field in the RPL DIOs [RFC6550] to its IPv6 address. The pledge learns the address of the JRC once joined and upon the reception of the first RPL DIO message, and uses it to operate as a JP.

7.1. Identification of Join Request Traffic

The join request traffic that is proxied by the Join Proxy (JP) comes from unauthenticated nodes, and there may be an arbitrary amount of it. In particular, an attacker may send fraudulent traffic in attempt to overwhelm the network.

When operating as part of a [RFC8180] 6TiSCH minimal network using distributed scheduling algorithms, the join request traffic present may cause intermediate nodes to request additional bandwidth. An attacker could use this property to cause the network to overcommit bandwidth (and energy) to the join process.

The Join Proxy is aware of what traffic is join request traffic, and so can avoid allocating additional bandwidth itself. The Join Proxy SHOULD implement a bandwidth cap on outgoing join request traffic. This cap will not protect intermediate nodes as they can not tell

join request traffic from regular traffic. Despite the bandwidth cap implemented separately on each Join Proxy, the aggregate join request traffic from many Join Proxies may cause intermediate nodes to decide to allocate additional cells. It is undesirable to do so in response to the join request traffic. In order to permit the intermediate nodes to avoid this, the traffic needs to be tagged.

[RFC2597] defines a set of per-hop behaviors that may be encoded into the Diffserv Code Points (DSCPs). The Join Proxy SHOULD set the DSCP of join request packets that it produces as part of the relay process to AF43 code point (See Section 6 of [RFC2597]).

A Join Proxy that does not set the DSCP on traffic forwarded should set it to zero so that it is compressed out.

A Scheduling Function (SF) running on 6TiSCH nodes SHOULD NOT allocate additional cells as a result of traffic with code point AF43. Companion SF documents SHOULD specify how this recommended behavior is achieved.

7.2. Identification of Join Response Traffic

The JRC SHOULD set the DSCP of join response packets addressed to the Join Proxy to AF42 code point. Join response traffic can not be induced by an attacker as it is generated only in response to legitimate pledges (see Section 9.1.3). AF42 has lower drop probability than AF43, giving join response traffic priority in buffers over join request traffic.

Due to the convergecast nature of the DODAG, the 6LBR links are often the most congested, and from that point down there is progressively less (or equal) congestion. If the 6LBR paces itself when sending join response traffic then it ought to never exceed the bandwidth allocated to the best effort traffic cells. If the 6LBR has the capacity (if it is not constrained) then it should provide some buffers in order to satisfy the Assured Forwarding behavior.

Companion SF documents SHOULD specify how traffic with code point AF42 is handled with respect to cell allocation.

8. Application-level Configuration

The CoJP join exchange in Figure 1 is carried over CoAP [RFC7252] and the secure channel provided by OSCORE [I-D.ietf-core-object-security]. The (6LBR) pledge plays the role of a CoAP client; the JRC plays the role of a CoAP server. The JP implements CoAP forward proxy functionality [RFC7252]. Because the JP can also be a constrained device, it cannot implement a cache. If

the JP used the stateful CoAP proxy defined in [RFC7252], it would be prone to Denial-of-Service (DoS) attacks, due to its limited memory. Rather, the JP processes forwarding-related CoAP options and makes requests on behalf of the pledge, in a stateless manner by using the Stateless-Proxy option defined in this document.

The pledge designates a JP as a proxy by including the Proxy-Scheme option in CoAP requests it sends to the JP. The pledge also includes in the requests the Uri-Host option with its value set to the well-known JRC's alias, as specified in Section 9.1.1.

The JP resolves the alias to the IPv6 address of the JRC that it learned when it acted as a pledge, and joined the network. This allows the JP to reach the JRC at the network layer and forward the requests on behalf of the pledge.

The JP MUST add a Stateless-Proxy option to all the requests that it forwards on behalf of the pledge as part of the join process.

The value of the Stateless-Proxy option is set to the internal JP state, needed to forward the Join Response message to the pledge. The Stateless-Proxy option handling is defined in Section 10.

The JP also tags all packets carrying the Join Request message at the network layer, as specified in Section 7.1.

8.1. OSCORE Security Context

Before the (6LBR) pledge and the JRC may start exchanging CoAP messages protected with OSCORE, they need to derive the OSCORE security context from the parameters provisioned out-of-band, as discussed in Section 4.

The OSCORE security context MUST be derived as per Section 3 of [I-D.ietf-core-object-security].

- o the Master Secret MUST be the PSK.
- o the Master Salt MUST be empty.
- o the ID of the pledge MUST be set to the byte string 0x00. This identifier is used as the OSCORE Sender ID in the security context derivation, as the pledge initially plays the role of a CoAP client.
- o the ID of the JRC MUST be set to the byte string 0x4a5243 ("JRC" in ASCII). This identifier is used as the OSCORE Recipient ID in

the security context derivation, as the JRC initially plays the role of a CoAP server.

- o the ID Context MUST be set to the pledge identifier.
- o the Algorithm MUST be set to the value from [RFC8152], agreed out-of-band by the same mechanism used to provision the PSK. The default is AES-CCM-16-64-128.
- o the Key Derivation Function MUST be agreed out-of-band. Default is HKDF SHA-256 [RFC5869].

The derivation in [I-D.ietf-core-object-security] results in traffic keys and a common IV for each side of the conversation. Nonces are constructed by XOR'ing the common IV with the current sequence number and sender identifier. For details on nonce construction, refer to [I-D.ietf-core-object-security].

Implementations MUST ensure that multiple CoAP requests to different JRCs result in the use of the same OSCORE context, so that the sequence numbers are properly incremented for each request. The pledge typically sends requests to different JRCs if it is not provisioned with the network identifier and attempts to join one network at a time. A simple implementation technique is to instantiate the OSCORE security context with a given PSK only once and use it for all subsequent requests. Failure to comply will break the confidentiality property of the Authenticated Encryption with Associated Data (AEAD) algorithm due to the nonce reuse.

This OSCORE security context is used for initial joining of the (6LBR) pledge, where the (6LBR) pledge acts as a CoAP client, as well as for any later parameter updates, where the JRC acts as a CoAP client and the joined node as a CoAP server, as discussed in Section 9.2. A (6LBR) pledge is expected to have exactly one OSCORE security context with the JRC.

8.1.1. Persistency

Implementations MUST ensure that mutable OSCORE context parameters (Sender Sequence Number, Replay Window) are stored in persistent memory. A technique that prevents reuse of sequence numbers, detailed in Section 6.5.1 of [I-D.ietf-core-object-security], MUST be implemented. Each update of the OSCORE Replay Window MUST be written to persistent memory.

This is an important security requirement in order to guarantee nonce uniqueness and resistance to replay attacks across reboots and

rejoins. Traffic between the (6LBR) pledge and the JRC is rare, making security outweigh the cost of writing to persistent memory.

9. Constrained Join Protocol (CoJP)

Constrained Join Protocol (CoJP) is a lightweight protocol over CoAP [RFC7252] and a secure channel provided by OSCORE [I-D.ietf-core-object-security]. CoJP allows the (6LBR) pledge to request admission into a network managed by the JRC, and for the JRC to configure the pledge with the parameters necessary for joining the network, or advertising it in the case of 6LBR pledge. The JRC may update the parameters at any time, by reaching out to the joined node that formerly acted as a (6LBR) pledge. For example, network-wide rekeying can be implemented by updating the keying material on each node.

This section specifies how the CoJP messages are mapped to CoAP and OSCORE, CBOR data structures carrying different parameters, transported within CoAP payload, and the parameter semantics and processing rules.

CoJP relies on the security properties provided by OSCORE. This includes end-to-end confidentiality, data authenticity, replay protection, and a secure binding of responses to requests.

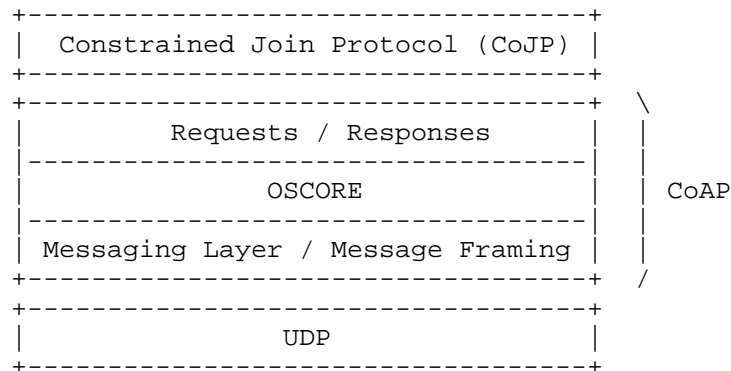


Figure 2: Abstract layering of CoJP.

When a (6LBR) pledge requests admission to a given network, it undergoes the CoJP join exchange that consists of:

- o the Join Request message, sent by the (6LBR) pledge to the JRC, potentially proxied by the JP. The Join Request message and its mapping to CoAP is specified in Section 9.1.1.

- o the Join Response message, sent by the JRC to the (6LBR) pledge if the JRC successfully processes the Join Request using OSCORE and it determines through a mechanism that is out of scope of this specification that the (6LBR) pledge is authorized to join the network. The Join Response message is potentially proxied by the JP. The Join Response message and its mapping to CoAP is specified in Section 9.1.2.

When the JRC needs to update the parameters of a joined node that formerly acted as a (6LBR) pledge, it executes the CoJP parameter update exchange that consists of:

- o the Parameter Update message, sent by the JRC to the joined node that formerly acted as a (6LBR) pledge. The Parameter Update message and its mapping to CoAP is specified in Section 9.2.1.
- o the Parameter Update Response message, sent by the joined node to the JRC in response to the Parameter Update message to signal successful reception of the updated parameters. The Parameter Update Response message and its mapping to CoAP is specified in Section 9.2.2.

The payload of CoJP messages is encoded with CBOR [RFC7049]. The CBOR data structures that may appear as the payload of different CoJP messages are specified in Section 9.3.

9.1. Join Exchange

This section specifies the messages exchanged when the (6LBR) pledge requests admission and configuration parameters from the JRC.

9.1.1. Join Request Message

The Join Request message SHALL be mapped to a CoAP request:

- o The request method is POST.
- o The type is Non-confirmable (NON).
- o The Proxy-Scheme option is set to "coap".
- o The Uri-Host option is set to "6tisch.arpa". This is an anycast type of identifier of the JRC that is resolved to its IPv6 address by the JP or the 6LBR pledge.
- o The Uri-Path option is set to "j".

- o The Object-Security option SHALL be set according to [I-D.ietf-core-object-security]. The OSCORE security context used is the one derived in Section 8.1. The OSCORE kid context is set to the ID context, which in turn is set to the pledge identifier. The OSCORE kid context allows the JRC to retrieve the security context for a given pledge.
- o The payload is a Join_Request CBOR object, as defined in Section 9.3.1.

9.1.2. Join Response Message

The Join Response message that the JRC sends SHALL be mapped to a CoAP response:

- o The response Code is 2.04 (Changed).
- o The payload is a Configuration CBOR object, as defined in Section 9.3.2.

9.1.3. Error Handling and Retransmission

Since the Join Request is mapped to a Non-confirmable CoAP message, OSCORE processing at the JRC will silently drop the request in case of a failure. This may happen for a number of reasons, including failed lookup of an appropriate security context (e.g. the pledge attempting to join a wrong network), failed decryption, positive replay window lookup, formatting errors (possibly due to malicious alterations in transit). Silently dropping the Join Request at the JRC prevents a DoS attack where an attacker could force the pledge to attempt joining one network at a time, until all networks have been tried.

Using a Non-confirmable CoAP message to transport the Join Request also helps minimize the required CoAP state at the pledge and the Join Proxy, keeping it to a minimum typically needed to perform CoAP congestion control. It does, however, introduce some complexity as the pledge needs to implement a retransmission mechanism.

The following binary exponential back-off algorithm is inspired by the one described in [RFC7252]. For each Join Request the pledge sends while waiting for a Join Response, the pledge MUST keep track of a timeout and a retransmission counter. For a new Join Request, the timeout is set to a random value between TIMEOUT_BASE and (TIMEOUT_BASE * TIMEOUT_RANDOM_FACTOR). The retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than MAX_RETRANSMIT, the Join Request is retransmitted, the retransmission counter is incremented, and the

timeout is doubled. Note that the retransmitted Join Request passes new OSCORE processing, such that the sequence number in the OSCORE context is properly incremented. If the retransmission counter reaches MAX_RETRANSMIT on a timeout, the pledge SHOULD attempt to join the next advertised 6TiSCH network. If the pledge receives a Join Response that successfully passes OSCORE processing, it cancels the pending timeout and processes the response. The pledge MUST silently discard any response not protected with OSCORE, including error codes. For default values of retransmission parameters, see Section 9.4.

If all join attempts to advertised networks have failed, the pledge SHOULD signal to the user the presence of an error condition, through some out-of-band mechanism.

9.2. Parameter Update Exchange

During the network lifetime, parameters returned as part of the Join Response may need to be updated. One typical example is the update of link-layer keying material for the network, a process known as rekeying. This section specifies a generic mechanism when this parameter update is initiated by the JRC.

At the time of the join, the (6LBR) pledge acts as a CoAP client and requests the network parameters through a representation of the "/j" resource, exposed by the JRC. In order for the update of these parameters to happen, the JRC needs to asynchronously contact the joined node. The use of the CoAP Observe option for this purpose is not feasible due to the change in the IPv6 address when the pledge becomes the joined node and obtains a global address.

Instead, once the (6LBR) pledge receives and successfully validates the Join Response and so becomes a joined node, it switches its CoAP role and becomes a server. The joined node exposes the "/j" resource that is used by the JRC to update the parameters. Consequently, the JRC operates as a CoAP client when updating the parameters. The request/response exchange between the JRC and the (6LBR) pledge happens over the already-established OSCORE secure channel.

9.2.1. Parameter Update Message

The Parameter Update message that the JRC sends to the joined node SHALL be mapped to a CoAP request:

- o The request method is POST.
- o The type is Confirmable (CON).

- o The Uri-Path option is set to "j".
- o The Object-Security option SHALL be set according to [I-D.ietf-core-object-security]. The OSCORE security context used is the one derived in Section 8.1. When a joined node receives a request with the Sender ID set to 0x4a5243 (ID of the JRC), it is able to correctly retrieve the security context with the JRC.
- o The payload is a Configuration CBOR object, as defined in Section 9.3.2.

The JRC has implicit knowledge on the global IPv6 address of the joined node, as it knows the pledge identifier that the joined node used when it acted as a pledge, and the IPv6 network prefix. The JRC uses this implicitly derived IPv6 address of the joined node to directly address CoAP messages to it.

9.2.2. Parameter Update Response Message

The Parameter Update Response message that the joined node sends to the JRC SHALL be mapped to a CoAP response:

- o The response Code is 2.04 (Changed).
- o The payload is empty.

9.3. CoJP Objects

This section specifies the structure of CoJP CBOR objects that may be carried as the payload of CoJP messages. Some of these objects may be received both as part of the CoJP join exchange when the device operates as a (CoJP) pledge, or the parameter update exchange, when the device operates as a joined (6LBR) node.

9.3.1. Join Request Object

The Join_Request structure is built on a CBOR map object.

The set of parameters that can appear in a Join_Request object is summarized below. The defined labels can be found below, the details of this registry are in section "CoJP Parameters" registry Section 13.2.

- o role: The identifier of the role that the pledge requests to play in the network once it joins, encoded as an unsigned integer. Possible values are specified in Table 1. This parameter MAY be included. In case the parameter is omitted, the default value of 0, i.e. the role "6TiSCH Node", MUST be assumed.

- o network identifier: The identifier of the network, as discussed in Section 3, encoded as a CBOR byte string. This parameter may appear both in the Join Request and in the Join Response. When present in the Join Request, it hints to the JRC the network that the pledge is requesting to join, enabling the JRC to manage multiple networks. The pledge obtains the value of the network identifier from the received EB frames. This parameter **MUST** be included in a Join_Request object if the role parameter is set to "6TiSCH Node". This parameter **MAY** be included if the role parameter is set to "6LBR". The inclusion of this parameter by the 6LBR pledge depends on whether the parameter was exchanged during the one-touch process, which in turn depends on the operational constraints.

The CDDL fragment that represents the text above for the Join_Request follows.

```
Join_Request = {
    ? 1 : uint           ; role
    ? 5 : bstr           ; network identifier
}
```

Name	Value	Description	Reference
6TiSCH Node	0	The pledge requests to play the role of a regular 6TiSCH node, i.e. non-6LBR node.	[[this document]]
6LBR	1	The pledge requests to play the role of 6LoWPAN Border Router (6LBR).	[[this document]]

Table 1: Role values.

9.3.2. Configuration Object

The Configuration structure is built on a CBOR map object. The set of parameters that can appear in a Configuration object is summarized below. The defined labels can be found below, the details of this registry are in section "CoJP Key Usage Registry" Section 13.3.

- o link-layer key set: An array encompassing a set of cryptographic keys and their identifiers that are currently in use in the network, or that are scheduled to be used in the future. The encoding of individual keys is described in Section 9.3.2.1. The link-layer key set parameter **MAY** be included in a Configuration

object. When present, the link-layer key set parameter MUST contain at least one key. How the keys are installed and used differs for the 6LBR and other nodes. When 6LBR receives this parameter, it MUST remove any old keys it has installed from the previous key set and immediately install and start using the new keys for all outgoing and incoming traffic. When a non-6LBR node receives this parameter, it MUST install the keys, use them for any incoming traffic matching the key identifier, but keep using the old keys for all outgoing traffic. A non-6LBR node accepts any frames for which it has keys: both old and new keys. Upon reception and successful security processing of a link-layer frame secured with a key from the new key set, a non-6LBR node MUST remove any old keys it has installed from the previous key set. From that moment on, a non-6LBR node MUST use the keys from the new key set for all outgoing traffic. In the case when the pledge is joining for the first time, before sending the first outgoing frame secured with a received key, the pledge needs to successfully complete the security processing of an incoming frame. To do so, the pledge can wait to receive a new frame or it can also store an EB frame that it used to find the JP and use it for immediate security processing upon reception of the key set. The described mechanism permits the JRC to provision the new key set to all the nodes while the network continues to use the existing keys. When the JRC is certain that all (or enough) nodes have been provisioned with the new keys, then the JRC updates the 6LBR. In the special case when the JRC is co-located with the 6LBR, it can simply trigger the sending of a new broadcast frame (e.g. EB), secured with a key from the new key set. The frame goes out with the new key, and upon reception and successful security processing of the new frame all receiving nodes will switch to the new active keys. Outgoing traffic from those nodes will then use the new key, which causes an update of additional peers, and the network will switch over in a flood-fill fashion.

- o link-layer short address: IEEE Std 802.15.4 short address assigned to the pledge. The short address structure is described in Section 9.3.2.2. The link-layer short address parameter MAY be included in a Configuration object. When a node receives this parameter as part of the Parameter Update message, it MUST update its link-layer short address to the one received.
- o JRC address: the IPv6 address of the JRC, encoded as a byte string, with the length of 16 bytes. If the length of the byte string is different than 16, the parameter MUST be discarded. If the JRC is not co-located with the 6LBR and has a different IPv6 address than the 6LBR, this parameter MUST be included. In the special case where the JRC is co-located with the 6LBR and has the same IPv6 address as the 6LBR, this parameter MAY be included. If

the JRC address parameter is not present in the Join Response, this indicates that the JRC has the same IPv6 address as the 6LBR. The joined node can then discover the IPv6 address of the JRC through network control traffic. See Section 7.

- o network identifier: the identifier of the network, as discussed in Section 3, encoded as a byte string. When present in the Join Response, this parameter is only valid when received by the 6LBR pledge. The parameter indicates to the 6LBR the value of the network identifier it should advertise at the link layer. This parameter MUST NOT be included in the Join Response if the role parameter from the corresponding Join Request indicated 0, i.e. the role "6TiSCH Node". In the case where the corresponding Join_Request object does not contain the network identifier parameter, this parameter MUST be included. When the corresponding Join_Request object does contain the network identifier parameter, this parameter MAY be included in the Configuration object. This may happen if the JRC decides to overwrite the network identifier provisioned during the one-touch process. The value of the network identifier parameter from the Configuration object SHOULD take precedence over the value provisioned during the one-touch process.
- o network prefix: the IPv6 network prefix, encoded as a byte string. The length of the byte string determines the prefix length. This parameter is only valid when received by the 6LBR pledge. The parameter indicates to the 6LBR the value of the IPv6 network prefix. This parameter MAY be included in the Join Response if the role parameter from the corresponding Join_Request object indicated 1, i.e. the role "6LBR". This parameter MUST NOT be included in the Join Response if the role parameter from the corresponding Join_Request object indicated 0, i.e. the role "6TiSCH Node".

The CDDL fragment that represents the text above for the Configuration follows. Structures Link_Layer_Key and Short_Address are specified in Section 9.3.2.1 and Section 9.3.2.2.

```
Configuration = {  
  ? 2 : [ +Link_Layer_Key ],      ; link-layer key set  
  ? 3 : Short_Address,            ; link-layer short address  
  ? 4 : bstr                      ; JRC address  
  ? 5 : bstr                      ; network identifier  
  ? 6 : bstr                      ; network prefix  
}
```

Name	Label	CBOR type	Description	Reference
role	1	unsigned integer	Identifies the role parameter.	[[this document]]
link-layer key set	2	array	Identifies the array carrying one or more link-level cryptographic keys.	[[this document]]
link-layer short address	3	array	Identifies the assigned link-layer short address	[[this document]]
JRC address	4	byte string	Identifies the IPv6 address of the JRC	[[this document]]
network identifier	5	byte string	Identifies the network identifier parameter	[[this document]]
network prefix	6	byte string	Identifies the IPv6 prefix of the network	[[this document]]

Table 2: Join Response map labels.

9.3.2.1. Link-Layer Key

The `Link_Layer_Key` structure encompasses the parameters needed to configure the link-layer security module: the value of the cryptographic key, the key identifier, the link-layer algorithm identifier, and the security level and the frame types that it should be used with, both for outgoing and incoming security operations.

For encoding compactness, `Link_Layer_Key` object is not enclosed in a top-level CBOR object. Rather, it is transported as a consecutive group of CBOR elements, with some being optional. To be able to decode the keys that are present in the link-layer key set, and to identify individual parameters of a single `Link_Layer_Key` object, the CBOR decoder needs to differentiate between elements based on the CBOR type. For example, when the decoder determines that the current element in the array is a byte string, it is certain that it is processing the last element of a given `Link_Layer_Key` object.

The set of parameters that can appear in a Link_Layer_Key object is summarized below, in order:

- o **key_index**: The identifier of the key, encoded as a CBOR unsigned integer. This parameter **MUST** be included. The parameter uniquely identifies the key and is used to retrieve the key for incoming traffic. In case of [IEEE802.15.4], the decoded CBOR unsigned integer value sets the "secKeyIndex" parameter that is signaled in all outgoing and incoming frames secured with this key. If the decoded CBOR unsigned integer value is larger than the maximum link-layer key identifier, which is 255 in [IEEE802.15.4]), the key is considered invalid. Additionally, in case of [IEEE802.15.4], the value of 0 is considered invalid. In case the key is considered invalid, the implementation **MUST** discard the key and attempt to decode the next key in the array.
- o **key_usage**: The identifier of the link-layer algorithm, security level and link-layer frame types that can be used with the key, encoded as a CBOR unsigned or negative integer. This parameter **MAY** be included. Possible values and the corresponding link-layer settings are specified in IANA "CoJP Key Usage" registry (Section 13.3). In case the parameter is omitted, the default value of 0 from Table 3 **MUST** be assumed.
- o **key_value**: The value of the cryptographic key, encoded as a byte string. This parameter **MUST** be included. If the length of the byte string is different than the corresponding key length for a given algorithm specified by the key_usage parameter, the key **MUST** be discarded and the decoder should attempt to decode the next key in the array.

The CDDL fragment that represents the text above for the Link_Layer_Key follows.

```
Link_Layer_Key = (
    key_index      : uint,
    ? key_usage    : uint / nint,
    key_value      : bstr,
)
```

Name	Value	Algorithm	Description	Reference
6TiSCH-K1K2-ENC-MIC-32	0	IEEE802154-AES-CCM-128	Use MIC-32 for EBs, ENC-MIC-32 for DATA	[[this document]]

				and ACKNOWLEDGMENT.	
6TiSCH-K1K2-ENC-MIC-64	1	IEEE802154-AES-CCM-128	Use MIC-64 for EBs, ENC-MIC-64 for DATA and ACKNOWLEDGMENT.	[[this document]]	
6TiSCH-K1K2-ENC-MIC-128	2	IEEE802154-AES-CCM-128	Use MIC-128 for EBs, ENC-MIC-128 for DATA and ACKNOWLEDGMENT.	[[this document]]	
6TiSCH-K1K2-MIC-32	3	IEEE802154-AES-CCM-128	Use MIC-32 for EBs, DATA and ACKNOWLEDGMENT.	[[this document]]	
6TiSCH-K1K2-MIC-64	4	IEEE802154-AES-CCM-128	Use MIC-64 for EBs, DATA and ACKNOWLEDGMENT.	[[this document]]	
6TiSCH-K1K2-MIC-128	5	IEEE802154-AES-CCM-128	Use MIC-128 for EBs, DATA and ACKNOWLEDGMENT.	[[this document]]	
6TiSCH-K1-MIC-32	6	IEEE802154-AES-CCM-128	Use MIC-32 for EBs.	[[this document]]	
6TiSCH-K1-MIC-64	7	IEEE802154-AES-CCM-128	Use MIC-64 for EBs.	[[this document]]	
6TiSCH-K1-MIC-128	8	IEEE802154-AES-CCM-128	Use MIC-128 for EBs.	[[this document]]	
6TiSCH-K2-MIC-32	9	IEEE802154-AES-	Use MIC-32	[[this d	

			CCM-128	for DATA and ACKNOWL EDGMENT.	ocument]]
6TiSCH-K2-MIC-64	10		IEEE802154-AES- CCM-128	Use MIC-64 for DATA and ACKNOWL EDGMENT.	[[this d ocument]]
6TiSCH-K2-MIC-12 8	11		IEEE802154-AES- CCM-128	Use MIC-128 for DATA and ACKNOWL EDGMENT.	[[this d ocument]]
6TiSCH-K2-ENC- MIC-32	12		IEEE802154-AES- CCM-128	Use ENC- MIC-32 for DATA and AC KNOWLEDGMEN T.	[[this d ocument]]
6TiSCH-K2-ENC- MIC-64	13		IEEE802154-AES- CCM-128	Use ENC- MIC-64 for DATA and AC KNOWLEDGMEN T.	[[this d ocument]]
6TiSCH-K2-ENC- MIC-128	14		IEEE802154-AES- CCM-128	Use ENC- MIC-128 for DATA and AC KNOWLEDGMEN T.	[[this d ocument]]

Table 3: Key Usage values.

9.3.2.2. Short Address

The `Short_Address` object represents an address assigned to the pledge that is unique locally in the network. It is encoded as a CBOR array object, containing, in order:

- o `address`: The assigned locally-unique address, encoded as a byte string. This parameter MUST be included. In case of [IEEE802.15.4], if the length of the byte string is different than 2, the address is considered invalid. In case of [IEEE802.15.4], the value of this parameter is used to set the short address of IEEE Std 802.15.4 module. In case the address is considered

invalid, the decoder MUST silently ignore the Short_Address object.

- o lease_time: The validity of the address in seconds after the reception of the CBOR object, encoded as a CBOR unsigned integer. This parameter MAY be included. The node MUST stop using the assigned short address after the expiry of the lease_time interval. It is up to the JRC to renew the lease before the expiry of the previous interval. The JRC updates the lease by executing the Parameter Update exchange with the node and including the Short_Address in the Configuration object, as described in Section 9.2. In case the address lease expires, the node SHOULD initiate a new join exchange, as described in Section 9.1. In case this parameter is omitted, the value of positive infinity MUST be assumed, meaning that the address is valid for as long as the node participates in the network.

The CDDL fragment that represents the text above for the Short_Address follows.

```
Short_Address = [
    address      : bstr,
    ? lease_time : uint
]
```

9.4. Parameters

CoJP uses the following parameters:

Name	Default Value
TIMEOUT_BASE	10 s
TIMEOUT_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4

The values of TIMEOUT_BASE, TIMEOUT_RANDOM_FACTOR, MAX_RETRANSMIT may be configured to values specific to the deployment. The default values have been chosen to accommodate a wide range of deployments, taking into account dense networks.

9.5. Mandatory to Implement Algorithms

The mandatory to implement AEAD algorithm for use with OSCORE is AES-CCM-16-64-128 from [RFC8152]. This is the algorithm used for securing IEEE Std 802.15.4 frames, and hardware acceleration for it is present in virtually all compliant radio chips. With this choice, CoAP messages are protected with an 8-byte CCM authentication tag, and the algorithm uses 13-byte long nonces.

The mandatory to implement hash algorithm is SHA-256 [RFC4231].

The mandatory to implement key derivation function is HKDF [RFC5869], instantiated with a SHA-256 hash.

10. Stateless-Proxy CoAP Option

The CoAP proxy defined in [RFC7252] keeps per-client state information in order to forward the response towards the originator of the request. This state information includes at least the CoAP token, the IPv6 address of the host, and the UDP source port number.

The Stateless-Proxy CoAP option (see Figure 3) allows the proxy to be entirely stateless. The proxy inserts this option in the request to carry the state information needed for relaying the response back to the client. The proxy still keeps some general state (e.g. for congestion control or request retransmission), but no per-client state.

The Stateless-Proxy CoAP option is critical, Safe-to-Forward, not part of the cache key, not repeatable and opaque. When processed by OSCORE, the Stateless-Proxy option is neither encrypted nor integrity protected.

No.	C	U	N	R	Name	Format	Length
TBD	x		x		Stateless-Proxy	opaque	1-255

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 3: Stateless-Proxy CoAP Option

Upon reception of a Stateless-Proxy option, the CoAP server MUST echo it in the response. The value of the Stateless-Proxy option is internal proxy state that is opaque to the server. For security reasons, the option value MUST be authenticated, MUST include a freshness indicator (e.g. a sequence number or timestamp) and MAY be encrypted. The proxy may use a COSE structure [RFC8152] to wrap the

state information as the value of the Stateless-Proxy option. The key used for encryption/authentication of the state information may be known only to the proxy.

Once the proxy has received the CoAP response with a Stateless-Proxy option present, it decrypts/authenticates it, checks the freshness indicator and constructs the response for the client, based on the information present in the option value.

Note that a CoAP proxy using the Stateless-Proxy option is not able to return a 5.04 Gateway Timeout Response Code in case the request to the server times out. Likewise, if the response to the proxy's request does not contain the Stateless-Proxy option, for example when the option is not supported by the server, the proxy is not able to return the response to the client, and the client eventually times out.

11. Security Considerations

This document recommends that the (6LBR) pledge and JRC are provisioned with unique PSKs. The nonce used for the Join Request and the Join Response is the same, but used under a different key. The design differentiates between keys derived for requests and keys derived for responses by different sender identifiers. Note that the address of the JRC does not take part in nonce or key construction. Even in the case of a misconfiguration in which the same PSK is used for several pledges, the keys used to protect the requests/responses from/towards different pledges are different, as they are derived using the pledge identifier as Master Salt. The PSK is still important for mutual authentication of the (6LBR) pledge and the JRC. Should an attacker come to know the PSK, then a man-in-the-middle attack is possible. The well-known problem with Bluetooth headsets with a "0000" pin applies here.

Being a stateless relay, the JP blindly forwards the join traffic into the network. A simple bandwidth cap on the JP prevents it from forwarding more traffic than the network can handle. This forces attackers to use more than one Join Proxy if they wish to overwhelm the network. Marking the join traffic packets with a non-zero DSCP allows the network to carry the traffic if it has capacity, but encourages the network to drop the extra traffic rather than add bandwidth due to that traffic.

The shared nature of the "minimal" cell used for the join traffic makes the network prone to DoS attacks by congesting the JP with bogus traffic. Such an attacker is limited by its maximum transmit power. The redundancy in the number of deployed JPs alleviates the issue and also gives the pledge a possibility to use the best

available link for joining. How a network node decides to become a JP is out of scope of this specification.

At the beginning of the join process, the pledge has no means of verifying the content in the EB, and has to accept it at "face value". In case the pledge tries to join an attacker's network, the Join Response message will either fail the security check or time out. The pledge may implement a temporary blacklist in order to filter out undesired EBs and try to join using the next seemingly valid EB. This blacklist alleviates the issue, but is effectively limited by the node's available memory. Bogus beacons prolong the join time of the pledge, and so the time spent in "minimal" [RFC8180] duty cycle mode.

12. Privacy Considerations

The join solution specified in this document relies on the uniqueness of the pledge identifier within the namespace managed by the JRC. This identifier is transferred in clear as an OSCORE kid context. The use of the globally unique EUI-64 as pledge identifier simplifies the management but comes with certain privacy risks. The implications are thoroughly discussed in [RFC7721] and comprise correlation of activities over time, location tracking, address scanning and device-specific vulnerability exploitation. Since the join protocol is executed rarely compared to the network lifetime, long-term threats that arise from using EUI-64 as the pledge identifier are minimal. In addition, the Join Response message contains a short address which is assigned by the JRC to the (6LBR) pledge. The assigned short address SHOULD be uncorrelated with the long-term pledge identifier. The short address is encrypted in the response. Once the join process completes, the new node uses the short addresses for all further layer 2 (and layer-3) operations. This mitigates the aforementioned privacy risks as the short layer-2 address (visible even when the network is encrypted) is not traceable between locations and does not disclose the manufacturer, as is the case of EUI-64.

13. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[this document]]" with the RFC number of this specification.

This document allocates a well-known name under the .arpa name space according to the rules given in [RFC3172]. The name "6tisch.arpa" is requested. No subdomains are expected. No A, AAAA or PTR record is requested.

13.1. CoAP Option Numbers Registry

The Stateless-Proxy option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Stateless-Proxy	\\[\\[this document\\]\\]

13.2. CoJP Parameters Registry

This section defines a sub-registries within the "IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) parameters" registry with the name "Constrained Join Protocol Parameters Registry".

The columns of the registry are:

Name: This is a descriptive name that enables an easier reference to the item. It is not used in the encoding.

Label: The value to be used to identify this parameter. The label is an unsigned integer.

CBOR type: This field contains the CBOR type for the field.

Description: This field contains a brief description for the field.

Reference: This field contains a pointer to the public specification for the field, if one exists.

This registry is to be populated with the values in Table 2.

The amending formula for this sub-registry is: Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

13.3. CoJP Key Usage Registry

This section defines a sub-registries within the "IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) parameters" registry with the name "Constrained Join Protocol Key Usage Registry".

The columns of this registry are:

Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.

Value: This is the value used to identify the key usage setting. These values MUST be unique. The value is an integer.

Algorithm: This is a descriptive name of the link-layer algorithm in use and uniquely determines the key length. The name is not used in the encoding.

Description: This field contains a description of the key usage setting. The field should describe in enough detail how the key is to be used with different frame types, specific for the link-layer technology in question.

References: This contains a pointer to the public specification for the field, if one exists.

This registry is to be populated with the values in Table 3.

The amending formula for this sub-registry is: Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

14. Acknowledgments

The work on this document has been partially supported by the European Union's H2020 Programme for research, technological development and demonstration under grant agreement No 644852, project ARMOUR.

The authors are grateful to Thomas Watteyne, Goeran Selander, Xavier Vilajosana, Pascal Thubert for reviewing, and to Klaus Hartke for providing input on the Stateless-Proxy CoAP option.

The authors would also like to thank Francesca Palombini, Ludwig Seitz and John Mattsson for participating in the discussions that have helped shape the document.

The IANA considerations for the three created registries is copied verbatim from RFC8392 at the suggestion of Mike Jones.

15. References

15.1. 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-13 (work in
progress), May 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](https://www.rfc-editor.org/info/rfc2119)>.
- [RFC2597] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski,
"Assured Forwarding PHB Group", RFC 2597,
DOI 10.17487/RFC2597, June 1999, <[https://www.rfc-
editor.org/info/rfc2597](https://www.rfc-editor.org/info/rfc2597)>.
- [RFC3172] Huston, G., Ed., "Management Guidelines & Operational
Requirements for the Address and Routing Parameter Area
Domain ("arpa")", BCP 52, RFC 3172, DOI 10.17487/RFC3172,
September 2001, <<https://www.rfc-editor.org/info/rfc3172>>.
- [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](https://www.rfc-editor.org/info/rfc7252)>.
- [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>>.

15.2. Informative References

- [I-D.ietf-6tisch-6top-protocol]
Wang, Q., Vilajosana, X., and T. Watteyne, "6top Protocol (6P)", draft-ietf-6tisch-6top-protocol-11 (work in progress), March 2018.
- [I-D.ietf-6tisch-architecture]
Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", draft-ietf-6tisch-architecture-14 (work in progress), April 2018.
- [I-D.ietf-6tisch-terminology]
Palattella, M., Thubert, P., Watteyne, T., and Q. Wang, "Terms Used in IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-terminology-10 (work in progress), March 2018.
- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.
- [IEEE802.15.4]
IEEE standard for Information Technology, ., "IEEE Std 802.15.4 Standard for Low-Rate Wireless Networks", n.d..
- [RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, DOI 10.17487/RFC4231, December 2005, <<https://www.rfc-editor.org/info/rfc4231>>.
- [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>>.
- [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>>.
- [RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <<https://www.rfc-editor.org/info/rfc6550>>.

- [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>>.
- [RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement", RFC 7554, DOI 10.17487/RFC7554, May 2015, <<https://www.rfc-editor.org/info/rfc7554>>.
- [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>>.
- [RFC8180] Vilajosana, X., Ed., Pister, K., and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration", BCP 210, RFC 8180, DOI 10.17487/RFC8180, May 2017, <<https://www.rfc-editor.org/info/rfc8180>>.

Appendix A. Example

Figure 4 illustrates a successful join protocol exchange. The pledge instantiates the OSCORE context and derives the traffic keys and nonces from the PSK. It uses the instantiated context to protect the Join Request addressed with a Proxy-Scheme option, the well-known host name of the JRC in the Uri-Host option, and its EUI-64 as pledge identifier and OSCORE kid context. Triggered by the presence of a Proxy-Scheme option, the JP forwards the request to the JRC and adds the Stateless-Proxy option with value set to the internally needed state. The JP has learned the IPv6 address of the JRC when it acted as a pledge and joined the network. Once the JRC receives the request, it looks up the correct context based on the kid context parameter. OSCORE data authenticity verification ensures that the request has not been modified in transit. In addition, replay protection is ensured through persistent handling of mutable context parameters.

Once the JP receives the Join Response, it authenticates the Stateless-Proxy option before deciding where to forward. The JP sets its internal state to that found in the Stateless-Proxy option, and forwards the Join Response to the correct pledge. Note that the JP does not possess the key to decrypt the CBOR object (configuration) present in the payload. The Join Response is matched to the Join Request and verified for replay protection at the pledge using OSCORE processing rules. In this example, the Join Response does not

contain the IPv6 address of the JRC, the pledge hence understands the JRC is co-located with the 6LBR.

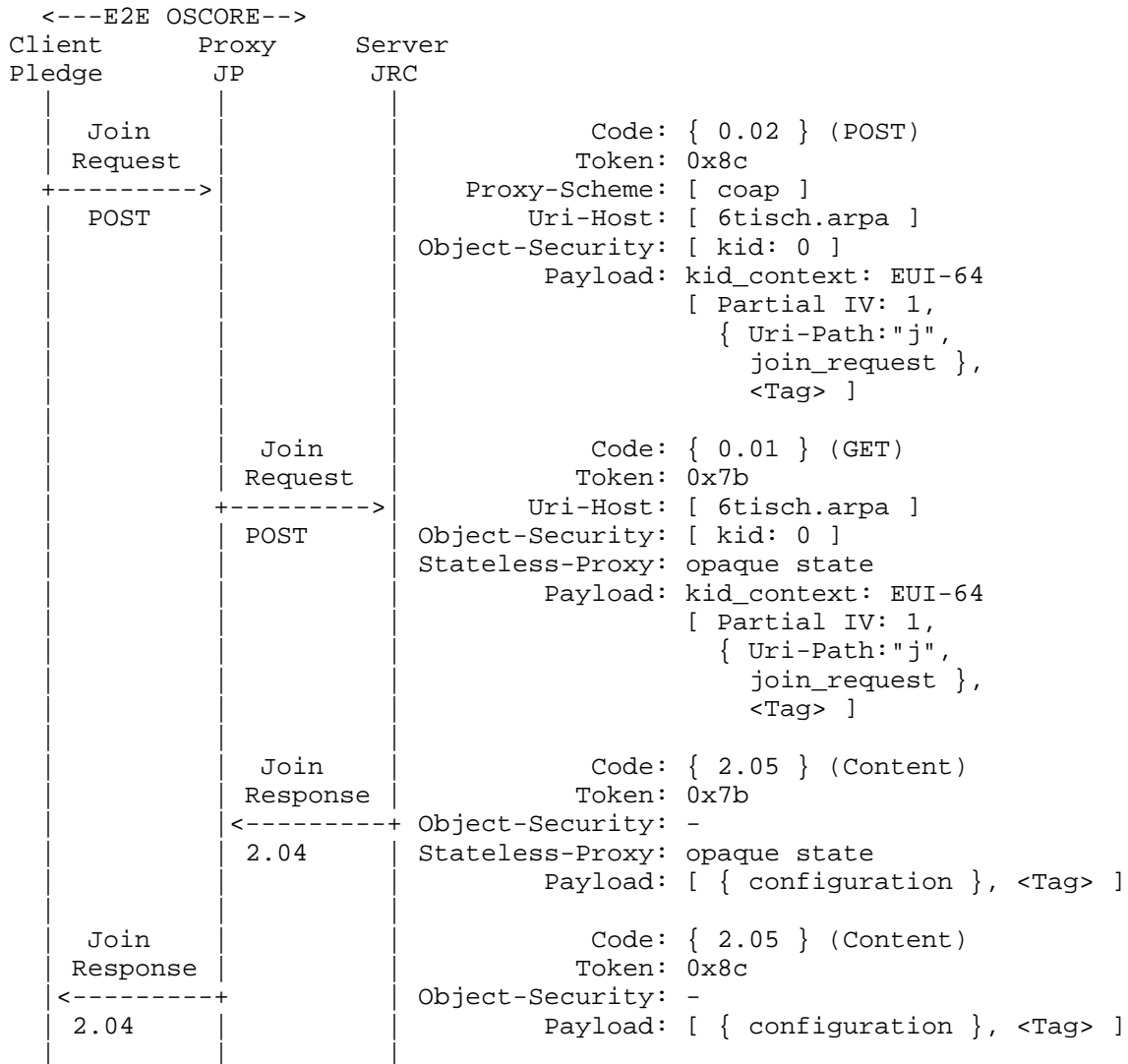


Figure 4: Example of a successful join protocol exchange. { ... } denotes encryption and authentication, [...] denotes authentication.

Where the join_request object is:

```
join_request:
{
    5 : h'cafe' / PAN ID of the network pledge is attempting to join /
}
```

Since the role parameter is not present, the default role of "6TiSCH Node" is implied.

The join_request object encodes to h'a10542cafe' with a size of 5 bytes.

And the configuration object is:

```
configuration:
{
    2 : [
        / link-layer key set /
        1, / key_index /
        h'e6bf4287c2d7618d6a9687445ffd33e6' / key_value /
    ],
    3 : [
        / link-layer short address /
        h'af93' / assigned short address /
    ]
}
```

Since the key_usage parameter is not present in the link-layer key set object, the default value of "6TiSCH-K1K2-ENC-MIC-32" is implied. Similarly, since the lease_time parameter is not present in the link-layer short address object, the default value of positive infinity is implied.

The configuration object encodes to

h'a202820150e6bf4287c2d7618d6a9687445ffd33e6038142af93' with a size of 26 bytes.

Authors' Addresses

Malisa Vucinic (editor)
University of Montenegro
Dzordza Vasingtona bb
Podgorica 81000
Montenegro

Email: malisav@ac.me

Jonathan Simon
Analog Devices
32990 Alvarado-Niles Road, Suite 910
Union City, CA 94587
USA

Email: jonathan.simon@analog.com

Kris Pister
University of California Berkeley
512 Cory Hall
Berkeley, CA 94720
USA

Email: pister@eecs.berkeley.edu

Michael Richardson
Sandelman Software Works
470 Dawson Avenue
Ottawa, ON K1Z5V7
Canada

Email: mcr+ietf@sandelman.ca

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
July 2, 2018

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-05

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 <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 January 3, 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. Terminology	3
3. Architecture	4
3.1. CoAP Pub/sub Architecture	4
3.2. CoAP Pub/sub Broker	4
3.3. CoAP Pub/sub Client	5
3.4. CoAP Pub/sub Topic	5
3.5. brokerless Pub/sub	5
4. CoAP Pub/sub REST API	6
4.1. DISCOVERY	6
4.2. CREATE	8
4.3. PUBLISH	11
4.4. SUBSCRIBE	13
4.5. UNSUBSCRIBE	15
4.6. READ	16
4.7. REMOVE	18
5. CoAP Pub/sub Operation with Resource Directory	19
6. Sleep-Wake Operation	20
7. Simple Flow Control	20
8. Security Considerations	20
9. IANA Considerations	21
9.1. Resource Type value 'core.ps'	22
9.2. Resource Type value 'core.ps.discover'	22
9.3. Response Code value '2.07'	22
10. Acknowledgements	22
11. References	22
11.1. Normative References	22
11.2. Informative References	23
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 or a client.

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.

This document specifies the means for nodes with limited reachability to communicate using simple extensions to CoAP. The extensions enable publish-subscribe communication using a Broker node that enables store-and-forward messaging between two or more nodes. Furthermore the extensions facilitate many-to-many communication using CoAP.

2. Terminology

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

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 topic 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.

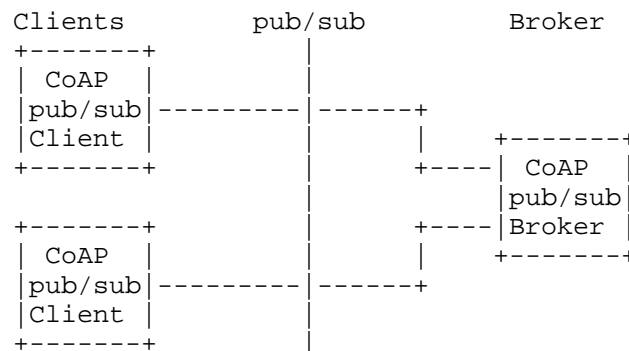


Figure 1: CoAP pub/sub Architecture

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://10.0.0.13:5684/EP-33543/sen/3303/0/5700"`.

3.5. brokerless Pub/sub

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.

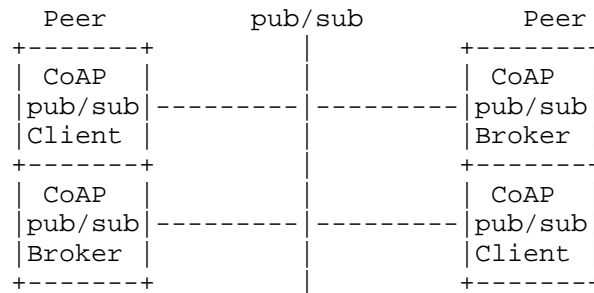


Figure 2: Brokerless pub/sub

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 `{create}`) 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 -----> -- Content-Format: application/link-format --- </pre>	<pre> <<--- 2.05 Content </ps/>;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" -----> Content-Format: application/link-format </pre>	<pre> <<--- 2.05 Content </ps/currentTemp>;rt="temperature";ct=50 --- </pre>

Figure 4: Example of DISCOVER topic

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

Figure 5: Example of DISCOVER topic

4.2. CREATE

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

the lifetime of the topic by including the Max-Age option in the CREATE request.

Topics may be created as sub-topics of other topics. A client MAY create a topic with a 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 Broker does not support the indicated format for both publish and subscribe, it MUST reject the operation with an error code of 4.00 "Bad Request".

There is no default content format. If no ct is specified, 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 return the URI path of the created topic via Location-Path options. The Broker MUST return the appropriate 4.xx response code indicating the reason for failure if a new topic can not be created. Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. 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.

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.

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.

Failure: 4.03 "Forbidden". Topic already exists.

Failure: 4.06 "Not Acceptable". Unsupported content format for topic.

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

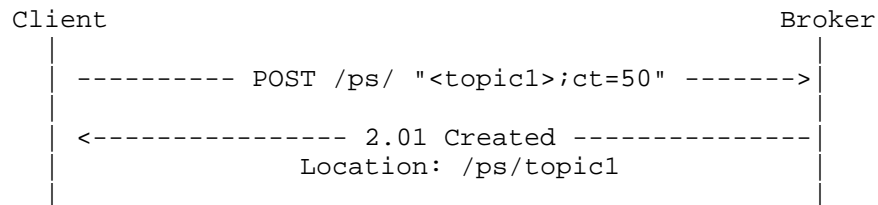


Figure 6: Example of CREATE topic

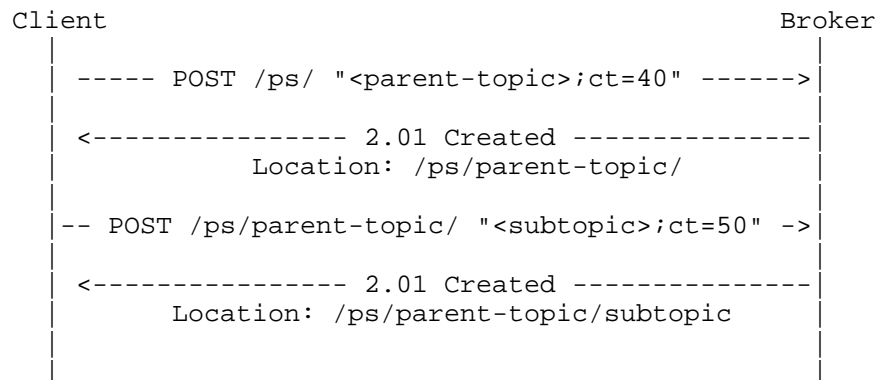


Figure 7: Example of CREATE of topic hierarchy

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 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.

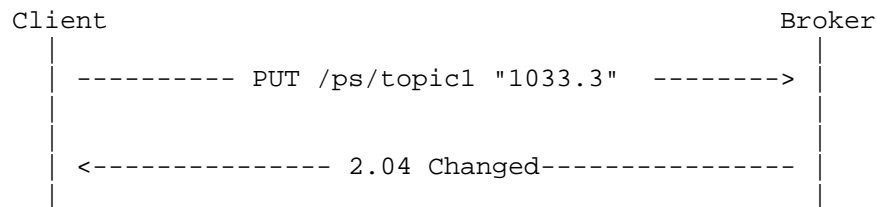


Figure 8: Example of PUBLISH

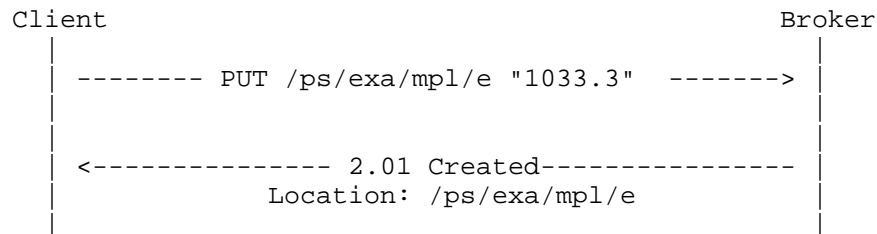


Figure 9: Example of CREATE on PUBLISH

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 of "2.07 No Content" if the topic has not yet been published to, or if Max-Age of the previously stored value has expired. The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed.

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

Success: 2.07 "No Content". Successful subscribe, 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.

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.

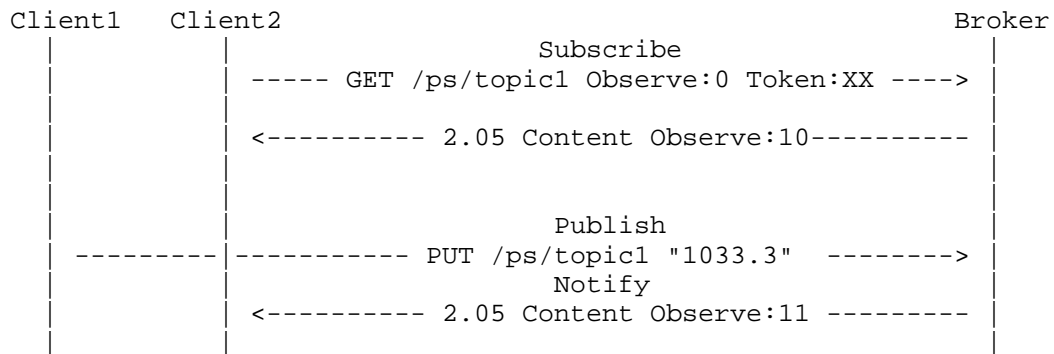


Figure 10: Example of SUBSCRIBE

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.

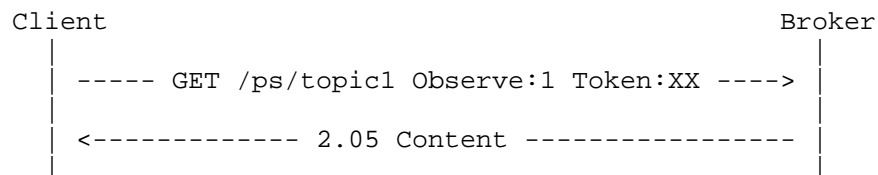


Figure 11: Example of UNSUBSCRIBE

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 topic to be valid since the last publish. The Broker MUST return a response code of "2.07 No Content" if the Max-Age of the previously stored value has expired, or if the topic has not yet been published to.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed. 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

Success: 2.07 "No Content". Topic exists, 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.

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.

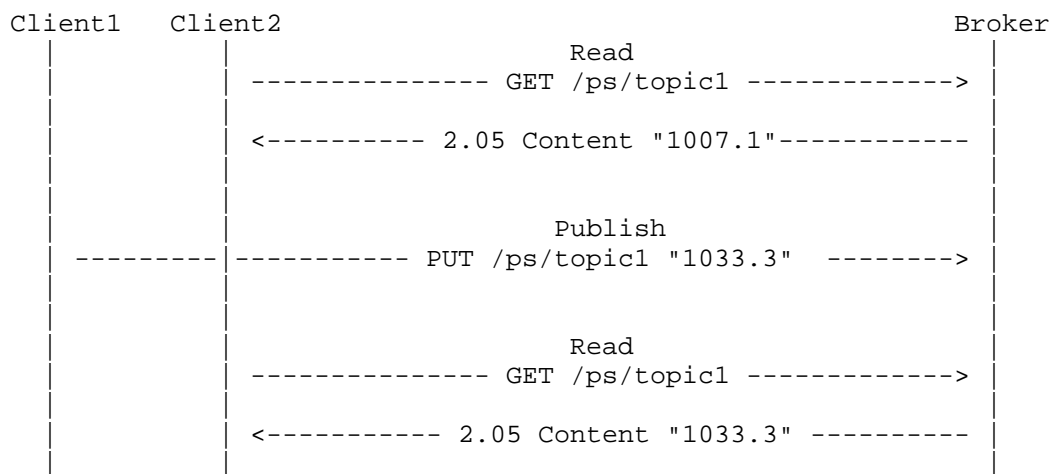


Figure 12: Example of READ

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.

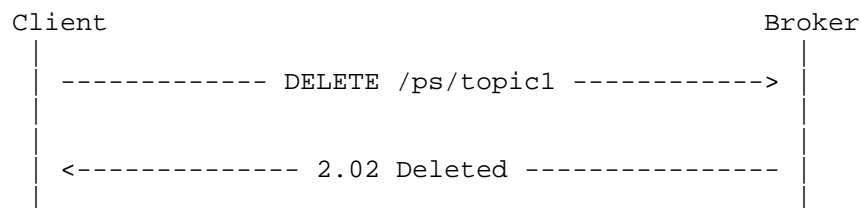


Figure 13: Example of REMOVE

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" [I-D.keranen-core-too-many-reqs] 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]. 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 receival, 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

9.3. Response Code value '2.07'

- o Response Code: 2.07
- o Description: No Content
- o Reference: [[This document]]
- o Notes: The server sends this code to the client to indicate that the request was valid and accepted, but the response may contain an empty payload. It is comparable to and may be proxied with the HTTP 204 No Content status code.

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

- [I-D.keranen-core-too-many-reqs]
Keranen, A., "Too Many Requests Response Code for the Constrained Application Protocol", draft-keranen-core-too-many-reqs-01 (work in progress), March 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>>.
- [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>>.

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-13 (work in progress), June 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-14 (work in progress), July 2018.
- [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>>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smarththings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: August 25, 2018

C. Bormann
Universitaet Bremen TZI
A. Betzler
Fundacio i2CAT
C. Gomez
I. Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
February 21, 2018

CoAP Simple Congestion Control/Advanced
draft-ietf-core-cocoa-03

Abstract

CoAP, the Constrained Application Protocol, needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. The CoRE CoAP specification defines basic behavior that exhibits low risk of congestion with minimal implementation requirements. It also leaves room for combining the base specification with advanced congestion control mechanisms with higher performance.

This specification defines more advanced, but still simple CoRE Congestion Control mechanisms, called CoCoA. The core of these mechanisms is a Retransmission TimeOut (RTO) algorithm that makes use of Round-Trip Time (RTT) estimates, in contrast with how the RTO is determined as per the base CoAP specification (RFC 7252). The mechanisms defined in this document have relatively low complexity, yet they improve the default CoAP RTO algorithm. The design of the mechanisms in this specification has made use of input from simulations and experiments in real networks.

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 25, 2018.

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	3
2. Context	4
3. Area of Applicability	4
4. Advanced CoAP Congestion Control: RTO Estimation	5
4.1. Blind RTO Estimate	6
4.2. Measurement-based RTO Estimate	6
4.2.1. Differences with the algorithm of RFC 6298	7
4.2.2. Discussion	7
4.3. Lifetime, Aging	8
5. Advanced CoAP Congestion Control: Non-Confirmables	9
5.1. Discussion	9
6. IANA Considerations	9
7. Security Considerations	10
8. References	10
8.1. Normative References	10
8.2. Informative References	11
Appendix A. Supporting evidence	11
A.1. Older versions of the draft and improvement	12
A.2. References	12
Appendix B. Pseudocode	13
B.1. Updating the RTO estimator	13
B.2. RTO aging	14
B.3. Variable Backoff Factor	14
Appendix C. Examples	15
C.1. Example A.1: weak RTTs	15
C.2. Example A.2: VBF and aging	15
C.3. Example B: VBF and aging	16
Appendix D. Analysis: difference between strong and weak	

estimators	16
Acknowledgements	17
Authors' Addresses	17

1. Introduction

CoAP, the Constrained Application Protocol, needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. The CoRE CoAP specification defines basic behavior that exhibits low risk of congestion with minimal implementation requirements. It also leaves room for combining the base specification with advanced congestion control mechanisms with higher performance.

The present specification defines such an advanced CoRE Congestion Control mechanism, with the goal of improving performance while retaining safety as well as the simplicity that is appropriate for constrained devices. Hence, we are calling this mechanism Simple Congestion Control/Advanced, or CoCoA for short.

CoCoA calculates the retransmission time-out (RTO) based on RTT estimations with and without loss. By taking retransmissions (in a potentially lossy network) into account when estimating the RTT, this algorithm reacts to congestion with a lower sending rate. For non-confirmable packets, it also limits the sending rate to $1/\text{RTO}$; assuming that the RTO estimation in CoCoA works as expected, RTO should be slightly greater than the RTT, thus CoCoA would be more conservative than the original specification in [RFC7641].

In the Internet, congestion control is typically implemented in a way that it can be introduced or upgraded unilaterally. Still, a new congestion control scheme must not be introduced lightly. To ensure that the new scheme is not posing a danger to the network, considerable work has been done on simulations and experiments in real networks. Some of this work will be mentioned in "Discussion" subsections in the following sections; an overview is given in Appendix A. Extended rationale for this specification can also be found in the historical Internet-Drafts [I-D.bormann-core-congestion-control] and [I-D.eggert-core-congestion-control], as well as in the minutes of the IETF 84 CoRE WG meetings.

1.1. Terminology

This specification uses terms from [RFC7252]. In addition, it defines the following terminology:

Initiator: The endpoint that sends the message that initiates an exchange. E.g., the party that sends a confirmable message, or a non-confirmable message (see Section 4.3 of [RFC7252]) conveying a request.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The term "byte", abbreviated by "B", is used in its now customary sense as a synonym for "octet".

2. Context

In the definition of the CoAP protocol [RFC7252], an approach was taken that includes a very simple basic scheme (lock-step with the number of parallel exchanges usually limited to 1) in the base specification together with performance-enhancing advanced mechanisms.

The present specification is based on the approved text in the [RFC7252] base specification. It is making use of the text that permits advanced congestion control mechanisms and allows them to change protocol parameters, including NSTART and the binary exponential backoff mechanism. Note that Section 4.8 of [RFC7252] limits the leeway that implementations have in changing the CoRE protocol parameters.

The present specification also assumes that, outside of exchanges, non-confirmable messages can only be used at a limited rate without an advanced congestion control mechanism (this is mainly relevant for [RFC7641]). It is also intended to address the [RFC8085] guideline about combining congestion control state for a destination; and to clarify its meaning for CoAP using the definition of an endpoint.

The present specification does not address multicast or dithering beyond basic retransmission dithering.

3. Area of Applicability

The present algorithm is intended to be generally applicable. The objective is to be "better" than default CoAP congestion control in a number of characteristics, including achievable goodput for a given offered load, latency, and recovery from bursts, while providing more predictable stress to the network and the same level of safety from catastrophic congestion. The algorithm defined in this document is

intended to adapt to the current characteristics of any underlying network, and therefore is well suited for a wide range of network conditions, in terms of bandwidth, latency, load, loss rate, topology, etc. In particular, CoCoA has been found to perform well in scenarios with latencies ranging from the order of milliseconds to peaks of dozens of seconds, as well as in single-hop and multihop topologies. Link technologies used in existing evaluation work comprise IEEE 802.15.4, GPRS, UMTS and Wi-Fi (see Appendix A). CoCoA is also expected to work suitably across the general Internet. The algorithm does require three state variables per scope plus the state needed to do RTT measurements, so it may not be applicable to the most constrained devices (say, class 1 as per [RFC7228]).

The scope of each instance of the algorithm in the current set of evaluations has been the five-tuple, i.e., CoAP + endpoint (transport address) for Initiator and Responder. Potential applicability to larger scopes needs to be examined.

4. Advanced CoAP Congestion Control: RTO Estimation

For an initiator that plans to make multiple requests to one destination endpoint, it may be worthwhile to make RTT measurements in order to compute a more appropriate RTO than the default initial timeout of 2 to 3 s. In particular, a wide spectrum of RTT values is expected in different types of networks where CoAP is used. Those RTTs range from several orders of magnitude below the default initial timeout to values larger than the default. The algorithm defined in this document is based on the algorithm for RTO estimation defined in [RFC6298], with appropriately extended default/base values, as proposed in Section 4.2.1. Note that such a mechanism must, during idle periods, decay RTO estimates that are shorter or longer than the default RTO estimate back to the default RTO estimate, until fresh measurements become available again, as proposed in Section 4.3.

RTT variability challenges RTO estimation. In TCP, delayed ACKs contribute to RTT variability, since this option adds a delay of up to 500 ms (typically, 200 ms) before an ACK is sent by a receiving TCP endpoint. However, one important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Also, for communications with networks of constrained devices that apply radio duty cycling, large and variable round-trip times are likely to be observed. Servers will only trigger their early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTO estimate shorter than 1 s SHOULD therefore use a larger backoff factor for retransmissions to avoid expending all of its retransmissions

(MAX_RETRANSMIT, see Section 4.2 of [RFC7252], normally 4) in the default interval of 2 to 3 s. The approach chosen for a mechanism with variable backoff factors is presented in Section 4.2.1.

It may also be worthwhile to perform RTT estimation not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be used to reduce the amount of state in an initiator is for further study.

4.1. Blind RTO Estimate

The initial RTO estimate for an endpoint is set to 2 seconds (the initial RTO estimate is used as the initial value for both E_weak_ and E_strong_ below).

If only the initial RTO estimate is available, the RTO estimate for each of up to NSTART exchanges started in parallel is set to 2 s times the number of parallel exchanges, e.g. if two exchanges are already running, the initial RTO estimate for an additional exchange is 6 seconds.

4.2. Measurement-based RTO Estimate

The RTO estimator runs two copies of the algorithm defined in [RFC6298], using the same variables and calculations to estimate the RTO, with the differences introduced in Section 4.2.1: One copy for exchanges that complete on initial transmissions (the "strong estimator", E_strong_), and one copy for exchanges that have run into retransmissions, where only the first two retransmissions are considered (the "weak estimator", E_weak_). For the latter, there is some ambiguity whether a response is based on the initial transmission or the retransmissions. For the purposes of the weak estimator, the time from the initial transmission counts. Responses obtained after the third retransmission are not used to update an estimator.

The overall RTO estimate is an exponentially weighted moving average computed of the strong and the weak estimator, which is evolved after each contribution to the weak estimator (1) or to the strong estimator (2), from the estimator (either the weak or strong estimator) that made the most recent contribution:

$$\text{RTO} := w_weak * E_weak_ + (1 - w_weak) * \text{RTO} \quad (1)$$

$$\text{RTO} := w_strong * E_strong_ + (1 - w_strong) * \text{RTO} \quad (2)$$

(Splitting this update into the two cases avoids making the contribution of the weak estimator too big in naturally lossy networks.)

The default values for the corresponding weights, `w_weak` and `w_strong`, are 0.25 and 0.5, respectively. These values have been found to offer good performance in evaluations (see Appendix A). Pseudocode and examples for the overall RTO estimate presented are available in Appendix B.1 and Appendix C.1.

4.2.1. Differences with the algorithm of RFC 6298

This subsection presents three differences of the algorithm defined in this document with the one defined in [RFC6298]. The first two recommend new parameter settings. The third one is the variable backoff factor (VBF), which replaces RFC6298's simple exponential backoff that always multiplies the RTO by a factor of 2 when the RTO timer expires.

The initial value for each of the two RTO estimators is 2 s.

For the weak estimator, the factor `K` (the RTT variance multiplier) is set to 1 instead of 4. This is necessary to avoid a strong increase of the RTO in the case that the `RTTVAR` value is very large, which may be the case if a weak RTT measurement is obtained after one or more retransmissions.

In order to avoid that exchanges with small initial RTOs (i.e. RTO estimate lower than 1 s) use up all retransmissions in a short interval of time, the RTO for a retransmission is multiplied by 3 for each retransmission as long as the RTO is less than 1 s.

On the other hand, to avoid exchanges with large initial RTOs (i.e., RTO estimate greater than 3 s) not being able to carry out all retransmissions within `MAX_TRANSMIT_WAIT` (normally 93 s), the RTO is multiplied only by 1.5 when RTO is greater than 3 s.

Pseudocode for the variable backoff factor is in Appendix B.3.

The binary exponential backoff is truncated at 32 seconds. Similar to the way retransmissions are handled in the base specification, they are dithered between `1 x RTO` and `ACK_RANDOM_FACTOR x RTO`.

4.2.2. Discussion

In contrast to [RFC6298], this algorithm attempts to make use of ambiguous information from retransmissions. This is motivated by the high non-congestion loss rates expected in constrained node networks,

and the need to update the RTO estimators even in the presence of loss. This approach appears to contravene the mandate in Section 3.1.1 of [RFC8085] that "latency samples MUST NOT be derived from ambiguous transactions". However, those samples are not simply combined into the strong estimator, but are used to correct the limited knowledge that can be gained from the strong RTT measurements by employing an additional weak estimator. In fact, the weak estimator allows to better update the RTO estimator when mostly weak RTTs are available, either due to the lossy nature of links or due to congestion-induced losses. In the presence of the latter, and compared to a strong-only estimator ($w_{\text{weak}}=0$), spurious timeouts are avoided and the rate of retries is reduced, which allows to decrease congestion. Evidence that has been collected from experiments appears to support that the overall effect of using this data in the way described is beneficial (Appendix A).

Some evaluation has been done on earlier versions of this specification [Betzler2013]. A more recent (and more comprehensive) reference is [Betzler2015].

4.3. Lifetime, Aging

The state of the RTO estimators for an endpoint SHOULD be kept as long as possible. If other state is kept for the endpoint (such as a DTLS connection), it is very strongly RECOMMENDED to keep the RTO state alive at least as long as this other state. In the absence of such other state, the RTO state SHOULD be kept at least long enough to avoid frequent returns to inappropriate initial values. For the default parameter set of Section 4.8 of [RFC7252], it is strongly RECOMMENDED to keep it for at least 255 s.

If an estimator has a value that is lower than 1 s, and it is left without further update for 16 times its current value, the RTO estimate is doubled. If an estimator has a value that is higher than 3 s, and it is left without further update for 4 times its current value, the RTO estimate is set to be

$$\text{RTO} := 1 \text{ s} + (0.5 * \text{RTO})$$

(Note that, instead of running a timer, it is possible to implement these RTO aging calculations cumulatively at the time the estimator is used next.)

Pseudocode and examples for the aging mechanism presented are available in Appendix B.2 and in Appendix C.2.

5. Advanced CoAP Congestion Control: Non-Confirmables

A CoAP endpoint MUST NOT send non-confirmables to another CoAP endpoint at a rate higher than defined by this document. Independent of any congestion control mechanisms, a CoAP endpoint can always send non-confirmables if their rate does not exceed 1 B/s.

Non-confirmables that form part of exchanges are governed by the rules for exchanges.

Non-confirmables outside exchanges (e.g., [RFC7641] notifications sent as non-confirmables) are governed by the following rules:

1. Of any 16 consecutive messages towards this endpoint that aren't responses or acknowledgments, at least 2 of the messages must be confirmable.
2. An RTO as specified in Section 4 must be used for confirmable messages.
3. The packet rate of non-confirmable messages cannot exceed $1/\text{RTO}$, where RTO is the overall RTO estimator value at the time the non-confirmable packet is sent.

5.1. Discussion

The mechanism defined above for non-confirmables is relatively conservative. More advanced versions of this algorithm could run a TFRC-style Loss Event Rate calculator [RFC5348] and apply the TCP equation to achieve a higher rate than $1/\text{RTO}$.

[RFC7641], Section 4.5.1, specifies that the rate of Non-Confirmables SHOULD NOT exceed $1/\text{RTT}$ on average, if the server can maintain an RTT estimate for a client. CoCoA limits the packet rate of Non-Confirmables in this situation to $1/\text{RTO}$. Assuming that the RTO estimation in CoCoA works as expected, $\text{RTO}[k]$ should be slightly greater than the $\text{RTT}[k]$, thus CoCoA would be more conservative. The expectation therefore is that complying with the NON rate set by CoCoA leads to complying with [RFC7641].

6. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

7. Security Considerations

The security considerations of, e.g., [RFC5681], [RFC2914], and [RFC8085] apply. Some issues are already discussed in the security considerations of [RFC7252].

If a malicious node manages to prevent the delivery of some packets, a consequence will be an RTO increase, which will further reduce network performance. Note that this type of attack is not specific for CoCoA (and not even specific for CoAP), and many congestion control algorithms increase the RTO upon packet loss detection. While it is hard to prevent radio jamming, some mitigation for other forms of this type of attack is provided by network access control techniques. Also, the weak estimator in CoCoA increases the chances of obtaining RTT measurements in the presence of heavy packet losses, allowing to keep the RTO updated, which in turn allows recovery from a jamming attack in reasonable time.

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>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [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>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [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

[Betzler2013]

Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "Congestion control in reliable CoAP communication", ACM MSWIM'13 p. 365-372, DOI 10.1145/2507924.2507954, 2013.

[Betzler2015]

Betzler, A., Gomez, C., Demirkol, I., and J. Paradells, "CoCoA+: an Advanced Congestion Control Mechanism for CoAP", Ad Hoc Networks Vol. 33 pp. 126-139, DOI 10.1016/j.adhoc.2015.04.007, October 2015.

[I-D.bormann-core-congestion-control]

Bormann, C. and K. Hartke, "Congestion Control Principles for CoAP", draft-bormann-core-congestion-control-02 (work in progress), July 2012.

[I-D.eggert-core-congestion-control]

Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.

[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>>.

[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. Supporting evidence

(Editor's note: The references local to this appendix may need to be merged with those from the specification proper, depending on the discretion of the RFC editor.)

CoCoA has been evaluated by means of simulation and experimentation in diverse scenarios comprising different link layer technologies, network topologies, traffic patterns and device classes. The main overall evaluation result is that CoCoA consistently delivers a performance which is better than, or at least similar to, that of default CoAP congestion control. While the latter is insensitive to network conditions, CoCoA is adaptive and makes good use of RTT samples.

It has been shown over real GPRS and IEEE 802.15.4 mesh network testbeds that in these settings, in comparison to default CoAP, CoCoA increases throughput and reduces the time it takes for a network to process traffic bursts, while not sacrificing fairness. In contrast, other RTT-sensitive approaches such as Linux-RTT or Peak-Hopper-RTT may be too simple or do not adapt well to IoT scenarios, underperforming default CoAP under certain conditions [1]. On the other hand, CoCoA has been found to reduce latency in GPRS and WiFi setups, compared with default CoAP [2].

CoCoA performance has also been evaluated for non-confirmable traffic over emulated GPRS/UMTS links and over a real IEEE 802.15.4 mesh testbed. Results show that since CoCoA is adaptive, it yields better packet delivery ratio than default CoAP (which does not apply congestion control to non-confirmable messages) or Observe (which introduces congestion control that is not adaptive to network conditions) [3, 4].

A.1. Older versions of the draft and improvement

CoCoA has evolved since its initial draft version. Its core has remained mostly stable since draft-bormann-core-cocoa-02. The evolution of CoCoA has been driven by research work. This process, including evaluations of early versions of CoCoA, as well as improvement proposals that were finally incorporated in CoCoA, is reflected in published works [5-10].

A.2. References

- [1] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP congestion control for the Internet of Things", IEEE Communications Magazine, July 2016.
- [2] F. Zheng, B. Fu, Z. Cao, "CoAP Latency Evaluation", draft-zheng-core-coap-lantency-evaluation-00, 2016 (work in progress).
- [3] A. Betzler, C. Gomez, I. Demirkol, "Evaluation of Advanced Congestion Control Mechanisms for Unreliable CoAP Communications", PE-WASUN, Cancun, Mexico, 2015.

- [4] A. Betzler, J. Isern, C. Gomez, I. Demirkol, J. Paradells, "Experimental Evaluation of Congestion Control for CoAP Communications without End-to-End Reliability", *Ad Hoc Networks*, Volume 52, 1 December 2016, Pages 183-194.
- [5] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "Congestion Control in Reliable CoAP Communication", 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'13), Barcelona, Spain, Nov. 2013.
- [6] A. Betzler, C. Gomez, I. Demirkol, M. Kovatsch, "Congestion Control for CoAP cloud services", 8th International Workshop on Service-Oriented Cyber-Physical Systems in Converging Networked Environments (SOCNE) 2014, Barcelona, Spain, Sept. 2014.
- [7] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoCoA+: an advanced congestion control mechanism for CoAP", *Ad Hoc Networks journal*, 2015.
- [8] Bhalerao, Rahul, Sridhar Srinivasa Subramanian, and Joseph Pasquale. "An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol." 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2016.
- [9] I Jaervinen, L Daniel, M Kojo, "Experimental evaluation of alternative congestion control algorithms for Constrained Application Protocol (CoAP)", IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015.
- [10] Balandina, Ekaterina, Yevgeni Koucheryavy, and Andrei Gurtov. "Computing the retransmission timeout in coap." *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer Berlin Heidelberg, 2013. 352-362.

Appendix B. Pseudocode

B.1. Updating the RTO estimator

```
// Default values
ALPHA = 0.125 // RFC 6298
BETA = 0.25 // RFC 6298
W_STRONG = 0.5
W_WEAK = 0.25

updateRTO(retransmissions, RTT) {
  if (retransmissions == 0) {
    RTTVAR_strong = (1 - BETA) * RTTVAR_strong
                  + BETA * (RTT_strong - RTT);
    RTT_strong = (1 - ALPHA) * RTT_strong + ALPHA * RTT;
    E_strong = RTT_strong + 4 * RTTVAR_strong;
    RTO = W_STRONG * E_strong + (1 - W_STRONG) * RTO;
  } else if (retransmissions <= 2) {
    RTTVAR_weak = (1 - BETA) * RTTVAR_weak
                 + BETA * (RTT_weak - RTT);
    RTT_weak = (1 - ALPHA) * RTT_weak + ALPHA * RTT;
    E_weak = RTT_weak + 1 * RTTVAR_weak;
    RTO = W_WEAK * E_weak + (1 - W_WEAK) * RTO;
  }
}
```

B.2. RTO aging

```
checkAging() {
  clock_time difference = getCurrentTime() - lastUpdatedTime;

  if ((RTO < 1s) && (difference > (16 * RTO))) {
    RTO = 2 * RTO;
    lastUpdatedTime = getCurrentTime();
  } else if ((RTO > 3s) && (difference > (4 * RTO))) {
    RTO = 1s + 0.5 * RTO;
    lastUpdatedTime = getCurrentTime();
  }
}
```

B.3. Variable Backoff Factor

```
backOffRTO() {
  if (RTO < 1s) {
    RTO = RTO * 3;
  } else if (RTO > 3s) {
    RTO = RTO * 1.5;
  } else {
    RTO = RTO * 2;
  }
}
```

Appendix C. Examples

C.1. Example A.1: weak RTTs

A large network of sensor nodes that report periodical measurements is operating normally, without congestion. The nodes transmit their sensor readings via CON messages every 20 s in an asynchronous way towards a server located behind a gateway, obtaining strong RTT measurements (RTT 1.1 s, RTTVAR 0.1 s) that lead to the calculation of an RTO of 1.5 s (in average) in each node. In this mode of operation, no aging is applied, since the RTO is refreshed before the aging mechanism applies.

Suddenly, upon detection of a global event, the majority of sensor nodes start transmitting at a higher rate (every 5 s) to increase the resolution of the acquired data, which creates heavy congestion that leads to packet losses and an important increase of real RTT between the nodes and the server (RTT 2 s, RTTVAR 1 s). Due to the packet losses and spurious retransmissions (which can fuel congestion even more), many nodes are not able to update their RTO via strong RTT measurements, but they are able to obtain weak RTT measurements. A node with an initial RTO of 1.5 s would run into a retransmission, before obtaining an ACK (given the RTT of 2 s and that the ACK is not lost).

This weak RTT measurement would increase the overall RTO of the node to 1.875 s ($RTO = 0.25 * 3 \text{ s} + 0.75 * 1.5 \text{ s}$). Following the same calculus (and RTT/RTTVAR values), after obtaining another weak RTT, the RTO would increase to 2.156 s. At this point, the benefits of the weak RTT measurements are twofold:

1. Further spurious retransmissions are avoided as the RTO has increased above the real RTT.
2. The increase of RTOs across the whole network reduces the rate with which retransmissions are generated, decreasing the network congestion (which leads to an RTT and packet loss decrease).

C.2. Example A.2: VBF and aging

Assuming that the frequency of message generation is even higher (every 3 s) and the real RTT would further increase due to congestion, the RTO at some point would increase to 4 s. Since now the RTO is above 3 s, no longer a binary backoff is used to avoid the RTO growing too much in case of retransmissions. As the generation of data from the nodes ceases at some point (the network returns to a normal state), the aging mechanism would reduce the RTO automatically

(with an RTO of 4 s, after 16 s the RTO would be shifted to 3 s before a new RTT is measured).

C.3. Example B: VBF and aging

A network of nodes connected over 4G with an Internet service is calculating very small RTO values (0.3 s) and the nodes are transmitting CON messages every 1 s. Suddenly, the connection quality gets worse and the nodes switch to a more stable, yet slower connection via GPRS. As a result of this change, the nodes run into retransmissions, as the real RTT has increased above the calculated RTO.

Since the RTO is below 1 s, the Variable Backoff Factor increases the backoff values quickly to avoid spurious retransmissions (0.9 s first retry, 2.7 s second retry, etc.). Further, if due to the packet losses and increased delays in the network no new RTT measurements are obtained, the aging mechanism automatically increases the RTO (doubling it) after 3.8 s ($16 * 0.3$ s) to adapt better to the sudden changes of network conditions. Without the Variable Backoff Factor and the aging mechanism, the number of spurious retransmissions would be much higher and the RTO would be corrected more slowly.

Appendix D. Analysis: difference between strong and weak estimators

This section analyzes the difference between the strong and weak RTO estimators. If there is no congestion, assume a static RTT of R' . Then, $E_strong_$ can be expressed as:

$$E_strong_ = R' + G,$$

since RTTVAR is reduced constantly by $RTTVAR = RTTVAR * 3/4$ (according to [RFC6298], and $SRTT=R'$), G would be dominant term in the $\max(G, K * RTTVAR)$ expression in the long run.

For the weak estimator: assume that the RTO setting converges to $E_strong_$ calculated above in the long run. If there is a packet loss, and an RTT is obtained for the first retransmission, then the weak RTT sample obtained by the weak estimator is:

$$RW' = R' + G + R'$$

Therefore, $E_weak_$ can be expressed as:

$$E_weak_ = RW' + \max(G, RW'/2) = 3 * R'$$

Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions. Several Transport Area representatives made further significant inputs this discussion during IETF84, including Lars Eggert, Michael Scharf, and David Black. Andrew McGregor, Eric Rescorla, Richard Kelsey, Ed Beroaset, Jari Arkko, Zach Shelby, Matthias Kovatsch and many others provided very useful additions. Further reviews by Michael Scharf and Ingemar Johansson led to further improvements, including some more discussion in the appendices.

Authors from Universitat Politecnica de Catalunya have been supported in part by the Spanish Government's Ministerio de Economia y Competitividad through projects TEC2009-11453, TEC2012-32531, TEC2016-79988-P and FEDER.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336. His contribution to this work has been carried out in part during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge, in collaboration with Prof. Jon Crowcroft.

Authors' Addresses

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

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

August Betzler
Fundacio i2CAT
Mobile and Wireless Internet Group
C/ del Gran Capita, 2
Barcelona 08034
Spain

Email: august.betzler@i2cat.net

Carles Gomez
Universitat Politecnica de Catalunya/Fundacio i2CAT
Escola d'Enginyeria de Telecomunicacio i Aeroespacial
de Castelldefels
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Phone: +34-93-413-7206
Email: carlesgo@entel.upc.edu

Ilker Demirkol
Universitat Politecnica de Catalunya/Fundacio i2CAT
Departament d'Enginyeria Telematica
C/Jordi Girona, 1-3
Barcelona 08034
Spain

Email: ilker.demirkol@entel.upc.edu

CoRE
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
June 04, 2018

CoAP Management Interface
draft-ietf-core-comi-03

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 core@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 December 6, 2018.

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	7
2.2. Compression of YANG identifiers	7
2.3. Instance identifier	8
2.4. CBOR ordered map schematic	9
2.5. Content-Formats	9
3. Example syntax	11
4. CoAP Interface	12
5. CoMI Collection Interface	13
5.1. Using the 'k' Uri-Query option	14
5.2. Data Retrieval	15
5.2.1. Using the 'c' Uri-Query option	16
5.2.2. Using the 'd' Uri-Query option	16
5.2.3. GET	17
5.2.4. FETCH	19
5.3. Data Editing	20
5.3.1. Data Ordering	20
5.3.2. POST	20
5.3.3. PUT	21
5.3.4. iPATCH	22
5.3.5. DELETE	23
5.4. Full datastore access	23
5.4.1. Full datastore examples	24
5.5. Event stream	25
5.5.1. Notify Examples	26
5.6. RPC statements	26
5.6.1. RPC Example	27
6. Access to MIB Data	27
7. Use of Block	29

8. Resource Discovery	29
9. Error Handling	31
10. Security Considerations	34
11. IANA Considerations	34
11.1. Resource Type (rt=) Link Target Attribute Values Registry	34
11.2. CoAP Content-Formats Registry	35
11.3. Media Types Registry	35
11.4. Concise Binary Object Representation (CBOR) Tags Registry	37
12. Acknowledgements	37
13. References	37
13.1. Normative References	38
13.2. 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	53
C.5. IP-MIB	54
Authors' Addresses	56

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 up to 10 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.

```
+-----+
| WARNING:                                     |
+-----+
| Mapping between data nodes and CoAP resources should be extended |
| in order to support the "Network Management Datastore           |
| Architecture" (NMDA) [RFC8342] and [RFC8342] and the YANG Schema |
| Mount [I-D.ietf-netmod-schema-mount].                          |
+-----+
```

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, configuration data, container, data model, data node, datastore, identity, instance identifier, key, key leaf, leaf, leaf-list, list, module, RPC, schema node, server, state data, submodule.

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.

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.

target resource: the resource that is associated with a particular CoAP request, identified by the request URI.

data node instance: An instance of a data node specified in a YANG module and stored in the server.

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 or 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.

data node value: The value assigned to a data node instance. Data 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.

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 [RFC7159] or XML [XML] as payload formats.
- o CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.
- o CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.
- 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 and RESTCONF also differ in the handling of:
 - * notifications.
 - * default values.

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

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 5.13.1. 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 5.1.

2.4. CBOR ordered map schematic

An ordered map is used as a root container of the application/yang-tree+cbor Content-Format. This datatype share the same functionalities as a CBOR map without the following limitations:

- o The ordering of the pairs of data items is preserved from serialization to deserialization.
- o Duplicate keys are allowed

This schematic is constructed using a CBOR array comprising pairs of data items, each pair consisting of a key that is immediately followed by a value. Unlike a CBOR map for which the length denotes the number of pairs, the length of the ordered map denotes the number of items (i.e. number of keys plus number of values).

The use of this schematic can be inferred from its context or by the presence of a preceding tag. The tag assigned to the Ordered map is defined in Section 11.4.

In the case of CoMI, the use of the ordered map as the root container of the application/yang-tree+cbor Content-Format is inferred, the Ordered map tag is not used.

2.5. Content-Formats

ComI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor]. All Content-Formats defined hereafter are constructed using one or both of these constructs:

- o YANG data node value, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 4.
- o YANG instance identifier, encoded based on the rules defined in [I-D.ietf-core-yang-cbor] section 5.13.1.

The following Content-formats are defined:

application/yang-value+cbor: represents a CBOR YANG document containing one YANG data node value. The YANG data node instance can be a leaf, a container, a list, a list instance, a RPC input, a RPC output, an action input, an action output, a leaf-list, an anydata or an anyxml. The CBOR encoding for each of these YANG data node instances are defined in [I-D.ietf-core-yang-cbor] section 4.

FORMAT: data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance, a RPC input, a RPC output, an action input, an actions output and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the URI carried in the associated request (i.e. GET, PUT, POST).

`application/yang-values+cbor`: represents a YANG document containing a list of data node values.

FORMAT: CBOR array of data-node-value

DELTA ENCODING: SIDs included in a YANG container, a list instance and an anydata are encoded using a delta value equal to the SID of the current schema node minus the SID of the parent. The parent SID of root data nodes is defined by the corresponding instance-identifier carried in the FETCH request.

`application/yang-tree+cbor`: represents a CBOR YANG document containing a YANG data tree.

FORMAT: ordered map of single-instance-identifier, data-node-value

DELTA ENCODING: The SID part of the first instance-identifier within the ordered map is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

`application/yang-selectors+cbor`: represents a CBOR YANG document containing a list of data node selectors (i.e. instance identifier).

FORMAT: CBOR array of instance-identifier

DELTA ENCODING: The SID part of the first instance-identifier within the CBOR array is encoded using its absolute value. Subsequent instance-identifiers are encoded using a delta value equal to the SID of the current instance-identifiers minus the SID of the previous instance-identifier.

`application/yang-patch+cbor`: represents a CBOR YANG document containing a list of data nodes to be replaced, created, or deleted.

For each data node instance, D, for which the instance identifier is the same as for a data node instance, I, in the targeted resource: the data node value of D replaces the data node value of

I. When the data node value of D is null, the data node instance I is removed. When the targeted resource does not contain a data node instance with the same instance identifier as D, a new data node instance is created in the targeted resource with the same instance identifier and data node value as D.

FORMAT: ordered map of instance-identifier, data-node-value

DELTA ENCODING: Same as Content-Format application/yang-tree+cbor

The different Content-formats usage is summarized in the table below:

Method	Resource	Content-Format
GET response	data node	/application/yang-value+cbor
PUT request	data node	/application/yang-value+cbor
POST request	data node	/application/yang-value+cbor
DELETE	data node	n/a
GET response	datastore	/application/yang-tree+cbor
PUT request	datastore	/application/yang-tree+cbor
POST request	datastore	/application/yang-tree+cbor
FETCH request	datastore	/application/yang-selectors+cbor
FETCH response	datastore	/application/yang-values+cbor
iPATCH request	datastore	/application/yang-patch+cbor
GET response	event stream	/application/yang-tree+cbor
POST request	rpc, action	/application/yang-value+cbor
POST response	rpc, action	/application/yang-value+cbor

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. The values of the SIDs are taken over from [yang-cbor].

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

The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Collection Interface. CoMI endpoints that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.datastore, with example path: /c, where c is short-hand for CoMI. The path /c is recommended but not compulsory (see Section 8).

Three CoMI resources are accessible with the following three example paths:

/c: Datastore resource with path "/c" and using CBOR content encoding format. Sub-resources of format /c/instance-identifier may be available to access directly each data node resource for this datastore.

/mod.uri: URI identifying the location of the YANG module library used by this server, with path "/mod.uri" and Content-Format "text/plain; charset=utf-8". An ETag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/s: Event stream resource to which YANG notification instances are reported. Notification support is optional, so this resource will not exist if the server does not support any notifications.

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/instance-identifier).

When multiple instances of a list exist, instance selection is possible as described in Section 5.1, Section 5.2.4, and Section 5.2.3.1.

The description of the management collection interface, with `if=core.c`, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

Function	Recommended path	rt
Datastore	/c	core.c.datastore
Data node	/c/instance-identifier	core.c.datanode
YANG module library	/mod.uri	core.c.moduri
Event stream	/s	core.c.eventstream

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

5. CoMI Collection Interface

The CoMI Collection Interface provides a CoAP interface to manage YANG servers.

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.

5.1. 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 5, item 8.

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

5.2. 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 7.

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.

5.2.1. 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".

5.2.2. 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'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable 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-value+cbor)
data-node-value

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

5.2.3.1. 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 octal 3273, yields two 6 bit decimal numbers 32 and 73, encoded in base64, (according to table 2 of [RFC4648]) yields a7. The response to the request returns

the CBOR encoding of this leaf of type 'string' as defined in [I-D.ietf-core-yang-cbor] section 5.4.

REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/yang-value+cbor)
"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-value+cbor)
{
 +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-value+cbor)
[
 {
 +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 /
 }
]

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 list key "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-value+cbor)
"Ethernet adaptor"

5.2.4. FETCH

The FETCH is used to retrieve multiple data node values. The FETCH request payload contains a list of instance-identifier encoded based on the rules defined by Content-Format application/yang-selectors+cbor in Section 2.5. The return response payload contains a list of values encoded based on the rules defined by Content-Format application/yang-values+cbor in Section 2.5. A value MUST be returned for each instance-identifier specified in the request. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

FORMAT:

FETCH /c (Content-Format :application/yang-selectors+cbor)
CBOR array of instance-identifier

2.05 Content (Content-Format: application/yang-values+cbor)
CBOR array of data-node-value

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix C.1. In the following 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-selectors+cbor)
[
  1723,                / current-datetime SID 1723 /
  [-190, "eth0"]       / interface SID 1533 with name = "eth0" /
]

RES:  2.05 Content (Content-Format :application/yang-value+cbor)
[
  "2014-10-26T12:16:31Z",    / current-datetime (SID 1723) /
  {
    +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) /
  }
]

```

5.3. Data Editing

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

5.3.1. 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.

5.3.2. 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 5.6 for details on "ACTION" and "RPC" resources.

A request to create a data node resource is sent with a confirmable 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-value+cbor)
  data-node-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

5.3.2.1. 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-value+cbor)
{
  +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

5.3.3. 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 confirmable CoAP PUT message.

```
FORMAT:
  PUT /c/<instanceidentifier>
      (Content-Format :application/yang-value+cbor)
  data-node-value

  2.01 Created
```

5.3.3.1. 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-value+cbor)
{
  +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

5.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent iPATCH method [RFC8132]. A request is sent with a confirmable 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-patch+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-patch+cbor)
ordered map of instance-identifier, data-node-value
```

2.04 Changed

5.3.4.1. 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 the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1756).

```

REQ: iPATCH /c (Content-Format :application/yang-patch+cbor)
[
  1755 , true,                                / enabled (1755) /
  [+1, "tac.nrc.ca"], null,                   / server (SID 1756) /
  +0,                                         / 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

5.3.5. DELETE

A data node resource is deleted with the DELETE method.

FORMAT:

Delete /c/<instance identifier>

2.02 Deleted

5.3.5.1. DELETE example

The example uses the interface list from Appendix C.3. Example is deleting an instance of the interface list (SID = 1533):

REQ: DELETE /c/X9?k="eth0"

RES: 2.02 Deleted

5.4. 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-tree+cbor)
ordered map of single-instance-identifier, data-node-value

FORMAT:

PUT /c (Content-Format: application/yang-tree+cbor)
ordered map of single-instance-identifier, data-node-value

2.04 Changed

FORMAT:

POST /c (Content-Format: application/yang-tree+cbor)
ordered map of single-instance-identifier, data-node-value

2.01 Created

FORMAT:

DELETE /c

2.02 Deleted

The content of the ordered map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request. When an Ordered map is used to carry a whole datastore, all data nodes MUST be identified using single instance identifiers (i.e. a SID), list instance identifiers are not allowed.

5.4.1. 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 list interface (SID 1533) with one instance and the container Clock (SID 1721). After invocation of GET, a map with these two modules is returned:

```

REQ:  GET /c

RES: 2.05 Content (Content-Format :application/yang-tree+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) /
  },
  -188,                                / clock (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) /
  }
]

```

5.5. 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 the implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero.

An example implementation is:

Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After appending the instance, the content of the instance is sent to all clients observing the modified stream.

Depending on the storage space allocated to the notification stream, the oldest notifications that do not fit inside the notification stream storage space are removed.

FORMAT:

```
Get /<stream-resource> Observe(0)
```

```
2.05 Content (Content-Format :application/yang-tree+cbor)
ordered map of instance-identifier, data-node-value
```

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

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

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

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

5.6. 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. Root data nodes are encoded using the delta between the current SID and the SID of the invoked instance identifier a specified by the URI.

The returned success response code is 2.05 Content.

FORMAT:

```
POST /c/<instance identifier>
      (Content-Format :application/yang-value+cbor)
data-node-value

2.05 Content (Content-Format :application/yang-value+cbor)
data-node-value
```

5.6.1. 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-value+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
}

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

6. 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, using diagnostic notation without delta encoding.

```

{
  60021 :                               / list ipNetToPhysicalEntry /
  [
    {
      60022 : 1,                        / ipNetToPhysicalIfIndex /
      60023 : 1,                        / ipNetToPhysicalNetAddressType /
      60024 : h'0A000033',             / ipNetToPhysicalNetAddress /
      60025 : h'00000A01172D',         / ipNetToPhysicalPhysAddress /
      60026 : 2333943,                 / ipNetToPhysicalLastUpdated /
      60027 : 4,                       / ipNetToPhysicalType /
      60028 : 1,                       / ipNetToPhysicalState /
      60029 : 1                        / ipNetToPhysicalRowStatus /
    },
    {
      60022 : 1,                        / ipNetToPhysicalIfIndex /
      60023 : 1,                        / ipNetToPhysicalNetAddressType /
      60024 : h'09020304',             / ipNetToPhysicalNetAddress /
      60025 : h'00000A36200A',         / ipNetToPhysicalPhysAddress /
      60026 : 2329836,                 / ipNetToPhysicalLastUpdated /
      60027 : 3,                       / ipNetToPhysicalType /
      60028 : 6,                       / ipNetToPhysicalState /
      60029 : 1                        / ipNetToPhysicalRowStatus /
    }
  ]
}

```

In this example one instance of /ip/ipNetToPhysicalEntry (SID 60021, base64: Oz1) that matches the keys ipNetToPhysicalIfIndex = 1, ipNetToPhysicalNetAddressType = ipv4 and ipNetToPhysicalNetAddress = 9.2.3.4 (h'09020304', base64: CQIDBA) is requested.

REQ: GET example.com/c/Oz1?k="1,1,CQIDBA"

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

```

{
  +1 : 1,                               / ( SID 60022 ) /
  +2 : 1,                               / ( SID 60023 ) /
  +3 : h'09020304',                     / ( SID 60024 ) /
  +4 : h'00000A36200A',                 / ( SID 60025 ) /
  +5 : 2329836,                         / ( SID 60026 ) /
  +6 : 3,                               / ( SID 60027 ) /
  +7 : 6,                               / ( SID 60028 ) /
  +8 : 1                                / ( SID 60029 ) /
}

```

7. 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 CBOR maps or CBOR arrays of undefined length, which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"core.c.datastore"` [RFC6690]. Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, the value `"/c"` is used as an example. The example below shows the discovery of the presence and location of management data.

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

```
RES: 2.05 Content
</c>; rt="core.c.datastore"
```

Implemented data nodes MAY be discovered using the standard CoAP resource discovery. The implementation can add the data node identifiers (SID) supported to `/.well-known/core` with

rt="core.c.datanode". The available SIDs can be discovered by sending a GET request to `"/.well-known/core"` including a resource type (rt) parameter with the value `"core.c.datanode"`. Upon success, the return payload will contain the registered SIDs and their location.

The example below shows the discovery of the presence and location of data nodes.

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

```
RES: 2.05 Content
```

```
</c/BaAiN>; rt="core.c.datanode",
```

```
</c/CF_fA>; rt="core.c.datanode"
```

The list of data nodes may become prohibitively long. Therefore, it is recommended to discover the details about the YANG modules implemented by reading a YANG module library (e.g. `"ietf-comi-yang-library"` as defined by `[I-D.veillette-core-yang-library]`).

The resource `"/mod.uri"` is used to retrieve the location of the YANG module library. This library can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

The following example shows the URI of a local instance of container modules-state (SID=1802) as defined in `[I-D.veillette-core-yang-library]`.

```
REQ: GET example.com/mod.uri
```

```
RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
```

```
example.com/c/cK
```

The following example shows the URI of a remote instance of same container.

```
REQ: GET example.com/mod.uri
```

```
RES: 2.05 Content (Content-Format: text/plain; charset=utf-8)
```

```
example-remote-server.com/group17/cK
```

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

9. 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 2.3.

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 delta value equal to the SID of the current schema node minus the SID of the parent container (i.e 1024).

```
+--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-value+cbor)
{
  +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) /
}
```

10. 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.

11. IANA Considerations

11.1. 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.datastore	YANG datastore	RFC XXXX
core.c.datanode	YANG data node	RFC XXXX
core.c.liburi	YANG module library	RFC XXXX
core.c.eventstream	YANG event stream	RFC XXXX

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

11.2. 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-value+cbor	XXX	RFC XXXX
application/yang-values+cbor	XXX	RFC XXXX
application/yang-selectors+cbor	XXX	RFC XXXX
application/yang-tree+cbor	XXX	RFC XXXX
application/yang-ipatch+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.

11.3. Media Types Registry

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

Name	Template	Reference
yang-value+cbor	application/yang-value+cbor	RFC XXXX
yang-values+cbor	application/yang-values+cbor	RFC XXXX
yang-selectors+cbor	application/yang-selectors+cbor	RFC XXXX
yang-tree+cbor	application/yang-tree+cbor	RFC XXXX
yang-ipatch+cbor	application/yang-ipatch+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.

11.4. Concise Binary Object Representation (CBOR) Tags Registry

This document adds the following tags to the "Concise Binary Object Representation (CBOR) Tags" registry.

Tag	Data Item	Semantics	Reference
xxx	array	Oedered map	RFC XXXX

// RFC Ed.: replace xxx by the assigned Tag and remove this note. //
 RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

12. 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.

13. References

13.1. Normative References

- [I-D.ietf-core-sid]
Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-03 (work in progress), December 2017.
- [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-06 (work in progress), February 2018.
- [I-D.veillette-core-yang-library]
Veillette, M., "Constrained YANG Module Library", draft-veillette-core-yang-library-02 (work in progress), January 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>>.
- [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>>.

13.2. Informative References

- [I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-11 (work in progress), March 2018.
- [I-D.ietf-netmod-schema-mount]
Bjorklund, M. and L. Lhotka, "YANG Schema Mount", draft-ietf-netmod-schema-mount-10 (work in progress), April 2018.
- [netconfcentral]
YUMAworks, "NETCONF Central: library of YANG modules", Web <http://www.netconfcentral.org/modulelist>.
- [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>>.

- [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>>.
- [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>>.
- [XML] W3C, "Extensible Markup Language (XML)", Web <http://www.w3.org/xml>.
- [yang-cbor] Veillette, M., "yang-cbor Registry", Web <https://github.com/core-wg/yang-cbor/tree/master/registry/>.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2017-07-01.yang"
module ietf-comi {
  yang-version 1.1;

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

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
    <mailto:michel.veillette@trilliant.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 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)";
}

typedef date_and_time_b {
  type int64;
  description
    "Binary representation of a date and time. This value is
    encoded using a positive or negative value representing
    a number of seconds relative to 1970-01-01T00:00Z in UTC
    time (i.e. the epoch). Negative values represent a date
    and time before the epoch, positive values represent a
    date and time after the epoch.
    This representation is defined in [RFC 7049] section
    2.4.1. When implemented using CoMI, tag 0 is assumed
    and not encoded.";
  reference
    "[RFC 7049] Concise Binary Object Representation (CBOR)";
}

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.";
}

container error {
    presence "Error payload";

    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": "2017-07-01",
  "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": "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. The YANG modules are available from [netconfcentral]. 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 {
  // SID 1700
  container system {
    // SID 1717
    container clock {
      // SID 1738
      choice timezone {
        case timezone-name {
          leaf timezone-name {
            // SID 1739
            type timezone-name;
          }
        }
      }
    }
  }
}

```

```
    }
    case timezone-utc-offset {
      leaf timezone-utc-offset { // SID 1740
        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

Peter van der Stok (editor)
consultant

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

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

Email: a@ackl.io

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

Email: andy@yumaworks.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

J. Arkko
Ericsson
C. Jennings
Cisco
Z. Shelby
ARM
July 2, 2018

Uniform Resource Names for Device Identifiers
draft-ietf-core-dev-urn-02

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 January 3, 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	3
3.1. Purpose	4
3.2. Syntax	4
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	6
3.9. Revision Information	6
4. DEV URN Subtypes	7
4.1. MAC Addresses	7
4.2. 1-Wire Device Identifiers	7
4.3. Organization-Defined Identifiers	7
4.4. Organization Serial Numbers	8
4.5. Organization Product and Serial Numbers	8
5. Examples	8
6. Security Considerations	9
7. IANA Considerations	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
Appendix A. Changes from Previous Version	12
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], [I-D.ietf-core-senml]. 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 [I-D.ietf-core-senml] [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 / opsbody / 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 / "_" / pct-encoding
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 [I-D.ietf-core-senml]).

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

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.

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>>.
- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <<https://www.rfc-editor.org/info/rfc3972>>.
- [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>>.
- [RFC5612] Eronen, P. and D. Harrington, "Enterprise Number for Documentation Use", RFC 5612, DOI 10.17487/RFC5612, August 2009, <<https://www.rfc-editor.org/info/rfc5612>>.
- [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](https://www.rfc-editor.org/info/rfc7252)>.

[I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C.
Bormann, "Media Types for Sensor Measurement Lists
(SenML)", draft-ietf-core-senml-13 (work in progress),
March 2018.

[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-15 (work in progress),
January 2018.

Appendix A. Changes from Previous Version

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 [I-D.ietf-core-senml], 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, 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: January 4, 2019

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology
July 03, 2018

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-06

Abstract

For CoAP (RFC7252), Dynamic linking of state updates between resources, either on an endpoint or between endpoints, is defined with the concept of Link Bindings. This specification defines conditional observation 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 January 4, 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	3
3. Link Bindings	4
3.1. The "bind" attribute and Binding Methods	4
3.1.1. Polling	5
3.1.2. Observe	5
3.1.3. Push	6
3.2. Link Relation	6
4. Binding and Resource Observation Attributes	6
4.1. Minimum Period (pmin)	7
4.2. Maximum Period (pmax)	7
4.3. Change Step (st)	7
4.4. Greater Than (gt)	8
4.5. Less Than (lt)	8
4.6. Notification Band (band)	8
4.7. Attribute Interactions	9
5. Binding Table	10
6. Implementation Considerations	11
7. Security Considerations	11
8. IANA Considerations	12
8.1. Interface Description	12
8.2. Link Relation Type	12
9. Acknowledgements	13
10. Contributors	13
11. Changelog	13
12. References	15
12.1. Normative References	15
12.2. Informative References	15
Appendix A. Examples	15
A.1. Greater Than (gt) example	15
A.2. Greater Than (gt) and Period Max (pmax) example	16

Authors' Addresses	17
--------------------	----

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 a set of conditional Observe Attributes for use with Link Bindings and with the standalone 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. 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 the content 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.

3.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the web-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.

In order to use binding methods, this specification defines a special CoRE link attribute "bind". This is the identifier of 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 1: 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 2: Binding Method Summary

The description of a binding method must define 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.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

The binding methods are described in more detail below.

3.1.1. 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).

3.1.2. 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 string parameters (see Section 4).

3.1.3. Push

When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the binding condition attributes are satisfied for the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

3.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, named "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.

4. Binding and Resource Observation Attributes

In addition to "bind", this specification further defines Web link attributes allowing a fine-grained control of the type of state synchronization along with the conditions that trigger an update.

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 using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters.

In addition, the set of parameters are 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, as query parameters.

These query parameters MUST be treated as resources that are read using GET and updated using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be updated at the same time by including the values in the query string of a PUT. Before being updated, these parameters have no default value.

These attributes are defined below:

Attribute	Parameter	Value
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)
Greater Than	gt	xsd:decimal
Less Than	lt	xsd:decimal
Notification Band	band	xsd:boolean

Table 3: Binding Attributes Summary

4.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time to wait (in seconds) before triggering a new state synchronization (even if it has changed). In the absence of this parameter, the minimum period is up to the synchronization initiator. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

4.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time in seconds between two consecutive state synchronizations (regardless if it has changed). In the absence of this parameter, the maximum period is up to the synchronization initiator. 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).

4.3. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a new state synchronization (compared to the value of the previous synchronization). Upon reception of a query including the st attribute the current value (CurrVal) of the resource is set as the initial value (STinit). Once the resource value differs from the STinit value (i.e. $\text{CurrVal} \geq \text{STinit} + \text{ST}$ or $\text{CurrVal} \leq \text{STinit} - \text{ST}$) then a new state

synchronization occurs. STinit is then set to the state synchronization value and new state synchronizations are based on a change step against this value. 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 an atomic numeric value.

Note: Due to the state synchronization based update of STint it may result in that resource value received in two sequential state synchronizations differs by more than st.

4.4. Greater Than (gt)

When present, Greater Than indicates the upper limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value exceeds the specified upper limit value. The actual resource value is used for the synchronization rather than the gt value. If the value continues to rise, no new state synchronizations are generated as a result of gt. If the value drops below the upper limit value and then exceeds the upper limit then a new state synchronization is generated.

4.5. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a new state synchronization. State synchronization only occurs when the resource value is less than the specified lower limit value. The actual resource value is used for the synchronization rather than the lt value. If the value continues to fall no new state synchronizations are generated as a result of lt. If the value rises above the lower limit value and then drops below the lower limit then a new state synchronization is generated.

4.6. Notification Band (band)

The notification band attribute allows a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple state synchronizations. 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). State synchronization occurs 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). State synchronization occurs 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, two kinds of signaling bands are specified.

If a band is specified in which the value of gt is less than that of lt, in-band signaling occurs. State synchronization occurs whenever the resource value is between the notification band minimum and maximum or is equal to the notification band minimum or maximum.

On the other hand if the band is specified in which the value of gt is greater than that of lt, out-of-band signaling occurs. State synchronization occurs whenever the resource value is outside the notification band minimum and maximum or is equal to the notification band minimum or maximum.

4.7. Attribute Interactions

Pmin, pmax, st, gt and lt may be present in the same query. Parameters are not defined at multiple prioritization levels. Instead, the server state machine generates a notification whenever any of the parameter conditions are met, after which it performs a reset on all the requested conditions. State synchronization also occurs only once even if there are multiple conditions being met at the same time.

5. Binding Table

The Binding table is a special resource that gives access to the bindings on a endpoint. This section defines a REST interface for Binding table resources. The Binding table resource MUST support the Binding interface defined below. The interface supports the link-format type.

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 specification. Although this interface description is intended to be used with the CoRE Link Format, it is applicable for use in any REST interface definition.

The Methods column defines the REST methods supported by the interface, which are described in more detail below.

Interface	if=	Methods	Content-Formats
Binding	core.bnd	GET, POST, DELETE	link-format

Table 4: Binding Interface Description

The Binding interface is used to manipulate a binding table. A request with a POST method and a content format of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table. Individual entries may be deleted from the table by specifying the resource path in a DELETE request.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```
Req: POST /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"

Req: DELETE /bnd/a/light
Res: 2.04 Changed

Req: DELETE /bnd/
Res: 2.04 Changed
```

Figure 1: Binding Interface 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, Bbm_x=20) and BandB (Bbm_n=21, Bbm_x=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.

7. Security Considerations

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. Consequently, consideration has to be given to what kinds of security credentials the the state machine 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.

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this specification. 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.

8. IANA Considerations

8.1. Interface Description

The specification registers the "binding" CoRE interface description link target attribute value as per [RFC6690].

Attribute Value: core.bnd

Description: The binding interface is used to manipulate 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.

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-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: Formalised 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

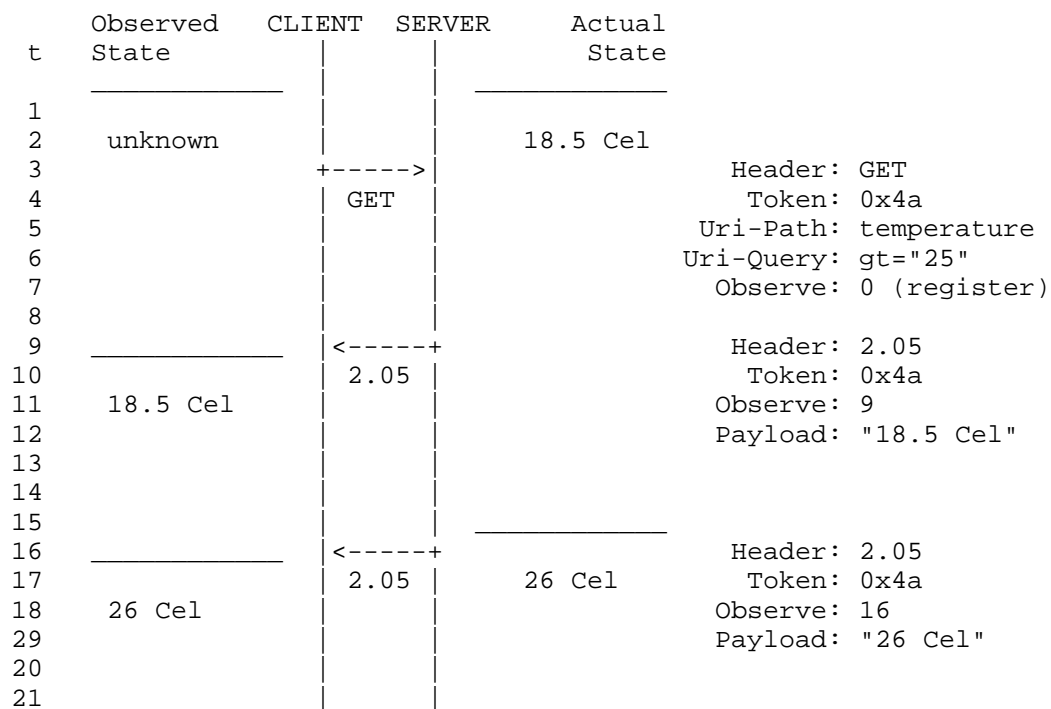
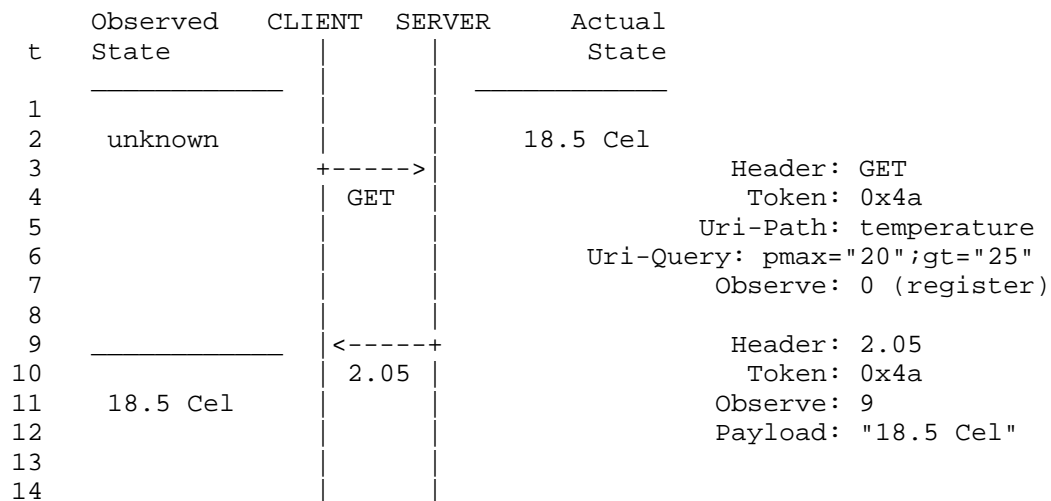


Figure 2: Client Registers and Receives one Notification of the Current State and One of a New State when it passes through the greather than threshold of 25.

A.2. Greater Than (gt) and Period Max (pmax) example



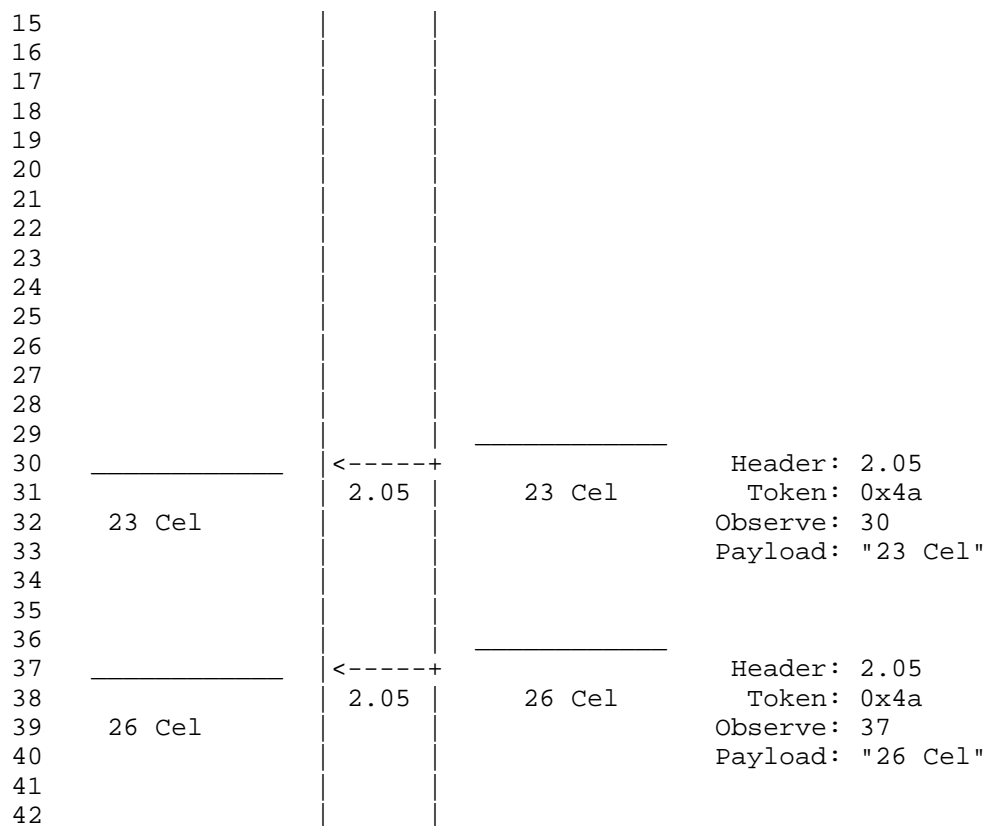


Figure 3: Client Registers and Receives one Notification of the Current State, one when pmax time expires and one of a new State when it passes through the greather than threshold of 25.

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 of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: December 31, 2018

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB
June 29, 2018

Echo and Request-Tag
draft-ietf-core-echo-request-tag-02

Abstract

This document specifies several 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 [RFC7252]. 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 December 31, 2018.

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	3
1.3. Request-Response Binding	4
1.4. Terminology	5
2. The Echo Option	5
2.1. Option Format	6
2.2. Echo Processing	6
2.3. Applications	9
3. The Request-Tag Option	10
3.1. Option Format	10
3.2. Request-Tag processing by servers	11
3.3. Setting the Request-Tag	12
3.4. Applications	12
3.4.1. Body Integrity Based on Payload Integrity	12
3.4.2. Multiple Concurrent Blockwise Operations	13
3.4.3. Simplified block-wise Handling for constrained proxies	14
3.5. Rationale for the option properties	14
3.6. Rationale for introducing the option	15
4. Block2 / ETag Processing	15
5. Token Processing	15
6. IANA Considerations	15
7. Security Considerations	16
8. References	17
8.1. Normative References	17
8.2. Informative References	17
Appendix A. Methods for Generating Echo Option Values	18
Appendix B. Request-Tag Message Size Impact	19
Appendix C. Change Log	20
Authors' Addresses	21

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 several security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two server-oriented CoAP options, the Echo option and the Request-Tag option, mainly addressing the security features request freshness and fragmented message body integrity, respectively. The Echo option enables a CoAP server to verify the freshness of a request, verify the aliveness of a client, 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 ([RFC5246]), 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.

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 the DTLS handshake) for each actuation, but in general this is not a solution suitable for constrained environments.

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 the 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. In HTTPS this 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 can return responses in any order. Concurrent requests are instead differentiated by their Token. 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 (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). 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 traffic 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 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 server-driven challenge-response mechanism for CoAP. The Echo option value is a challenge from the server to the

client included in a CoAP response and echoed in one or more CoAP request.

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	Length	Default	E
TBD			x		Echo	opaque	4-40	(none)	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, the following paragraph and the "E" column above need to move into OSCORE.]

The Echo option value is generated by the 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 the 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 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 also include the Echo option in a response to verify the aliveness of a client, to synchronize state, or 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 requirement. 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 $RTT = (t_1 - t_0)$, where t_1 is the request receive time and t_0 is the transmit time of the response that included the specific Echo option value. The server MUST only accept requests with a round-trip time below a certain threshold T , i.e. $RTT < T$, otherwise the request is not processed further, and an error message MAY be sent. 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.

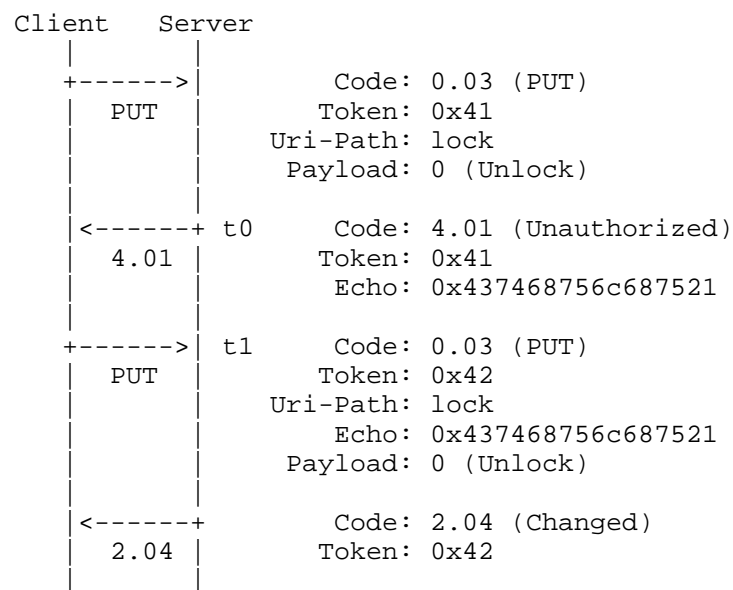


Figure 2: Example Echo Option Message Flow

When used to serve freshness requirements (including client aliveness and state synchronizing), CoAP requests containing the Echo option MUST be integrity protected, e.g. using DTLS, TLS, or OSCORE ([I-D.ietf-core-object-security]). When used to demonstrate reachability at their apparent network address, the Echo option MAY be used without protection.

Note that the server does not have to synchronize the time used for the Echo timestamps with any other party. If the server loses time synchronization, e.g. due to reboot, it MUST reject all Echo values that was created before time synchronization was lost.

CoAP-CoAP proxies MUST relay the Echo option unmodified. The CoAP server side of CoAP-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-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.
2. To avoid additional roundtrips for applications with multiple actuator requests in rapid sequence between the same client and server, the server may use 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.
3. If a server reboots during operation it may need to synchronize state with requesting clients 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.
4. When a device joins a multicast/broadcast group the device may need to synchronize state or time with the sender to ensure that the received message is fresh. By synchronizing time with the broadcaster, time can be used for synchronizing subsequent broadcast messages. 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.
5. 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, the 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.
6. A server may want to verify the aliveness of a client by responding with an Echo option.

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. 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	Length	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, the following paragraph and the "E"/"U" columns above need to move into OSCORE.]

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. 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-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-HTTP proxies and HTTP-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: As it varies the set of options that distinguish blockwise operations (ie. is neither Block1 or Block2 nor elective NoCacheKey), the server can not treat their messages 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 describes the individual blocks but not the operation as a whole. For example, a FETCH request with the same body can have a fresh response even if they were requested using different request tags. (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.

[The following paragraph might be better placed in lwig-coap, but was left here until lwig-coap has decided on its fate there.]

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.

Especially, that is the case for any correctly implemented proxy that does not know how to use Request-Tag in requests and has only one

client endpoint. When it receives concurrent incoming requests on the same resource, it needs to make that very choice: either send a 5.03 with Max-Age (holding off the second operation), or to commence the second operation and reject any further requests on the first operation with 4.08 Request Entity Incomplete errors without forwarding them. (Alternatively, it could spool the second request, but the unpredictable nature of the timeouts involved often makes that an unsuitable choice.)

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. 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.

When considering previous operations in protocols where the security association is not tightly bound to an end point (eg. OSCORE), the client MUST consider messages sent to `_any_` endpoint with 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.
[I-D.ietf-lwig-coap] Section TBD provides further details.

[Note to reviewers and co-authors: That section was so far only suggested in input for lwig-coap. If it does not get into the document, we should drop it here (for I don't want to explain all this case's details and security considerations here), but if the reference works, this section shows why Request-Tag has become repeatable.]

3.5. Rationale for the option properties

[This section needs to be reworked after assuming our RFC7959 interpretation.]

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.

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

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. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

Number	Name	Reference
TBD1	Echo	[RFC XXXX]
TBD2	Request-Tag	[RFC XXXX]

Figure 4: CoAP Option Numbers

7. Security 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 its wall clock time or location. Servers MUST use a monotonic clock to generate timestamps and compute round-trip times. Servers SHOULD NOT use wall clock time for timestamps, as wall clock time is not monotonic, may reveal that the server will accept expired certificates, or reveal the server's location. 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. 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 periodically. When resetting the timer, the server MUST reject all Echo values that was created before the reset.

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 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 minimizing state is to use the Integrity Protected Timestamp method described in Appendix A.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-13 (work in progress), June 2018.
- [I-D.ietf-lwig-coap]
Kovatsch, M., Bergmann, O., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-05 (work in progress), October 2017.

- [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-05 (work in progress), March
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>>.
- [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>>.

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. Use of one of the mechanisms outlined in this section is RECOMMENDED.

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.

- o Integrity Protected Timestamp. One method is to construct the Echo option value as 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 synchronization, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. A server MAY also want to encrypt its timestamps, 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

- o List of Cached Random Values and Timestamps. An alternative method is to construct the Echo option value as 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 synchronization, 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

A server MAY use different methods and security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

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.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: December 29, 2018

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University of Technology
June 27, 2018

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-12

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 December 29, 2018.

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	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	16
6.6. Read-only parameter	17
6.7. Actuator	17
7. Acknowledgements	17
8. Contributors	18
9. Changelog	18
10. References	21
10.1. Normative References	21
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)	23
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 [I-D.ietf-core-senml] 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)  
[  
  { "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 /l/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /l/
Res: 2.05 Content (application/senml+json)
[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" }
]
```

```
Req: POST /l/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /l/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /l/
Res: 2.05 Content (application/senml+json)
[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }
]
```

```
Req: DELETE /l/
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)
[
  { "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.

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.

8. Contributors

Matthieu Vial
Schneider-Electric
Grenoble
France

Phone: +33 (0)47657 6522
EMail: matthieu.vial@schneider-electric.com

9. Changelog

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., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-05 (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-13 (work in progress), March 2018.
- [I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", draft-ietf-core-senml-16 (work in progress), May 2018.
- [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>>.

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 of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 30, 2018

K. Li
Alibaba Group
A. Rahman
InterDigital
C. Bormann, Ed.
Universitaet Bremen TZI
February 26, 2018

Representing Constrained RESTful Environments (CoRE) Link Format in JSON
and CBOR
draft-ietf-core-links-json-10

Abstract

JavaScript Object Notation, JSON (RFC 8259) is a text-based data format which is popular for Web based data exchange. Concise Binary Object Representation, CBOR (RFC7049) is a binary data format which has been optimized for data exchange for the Internet of Things (IoT). For many IoT scenarios, CBOR formats will be preferred since it can help decrease transmission payload sizes as well as implementation code sizes compared to other data formats.

Web Linking (RFC 8288) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC 6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON, and similarly, inside constrained environments, in CBOR. This specification defines a common format for this.

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, 2018.

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. Objectives	3
1.2. Terminology	4
2. Web Links in JSON and CBOR	4
2.1. Background	4
2.2. Information Model	4
2.3. Additional Encoding Step for CBOR	6
2.4. Converting JSON or CBOR to Link-Format	8
2.5. Examples	9
2.5.1. Link Format to JSON Example	9
2.5.2. Link Format to CBOR Example	10
3. IANA Considerations	12
3.1. Media types	12
3.2. CoAP Content-Format Registration	13
4. Security Considerations	14
5. References	14
5.1. Normative References	14
5.2. Informative References	15
Appendix A. Reference implementation	16
Acknowledgements	19
Authors' Addresses	20

1. Introduction

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252].

The JavaScript Object Notation (JSON) [RFC8259] is a lightweight, text-based, language-independent data interchange format. JSON is popular in the Web development environment as it is easy for humans to read and write.

The Concise Binary Object Representation (CBOR) [RFC7049] is a binary data format which requires extremely small code size, allows very compact message representation, and provides extensibility without the need for version negotiation. CBOR is especially well suited for IoT environments because of these efficiencies.

When converting between a bespoke syntax such as that defined by [RFC6690] and JSON or CBOR, many small decisions have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge. This specification defines a common format for representing CoRE Web Linking in JSON and CBOR.

Note that there is a separate question on how to represent Web links pointing out of JSON documents, as discussed for example in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

1.1. Objectives

This specification has been designed based on the following objectives:

- o Canonical mapping
 - * lossless conversion in both directions between any pair of [RFC6690], JSON, and CBOR ("round-tripping"), unless prevented by a limitation of [RFC6690]
 - * but not attempting to ensure that a sequence of conversions from one of the formats through one or both of the others and back to the original would result in a bit-wise identical representation
- o The simplest thing that could possibly work.

While the formats defined in this document are based on the above objectives, they are general enough that they can be used for other applications of links in the Web. The same basic formats can be used for Web links that do not default to the "hosts" relation type (as is defined in [RFC6690]) and that allow percent encoding and general IRI syntax in what is an URI-Reference field in [RFC6690]. Also, specific support has been added for internationalized link attributes

such as "title*", including their language tags (while staying limited to UTF-8 as the character set).

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.

The term "byte" is used in its now customary sense as a synonym for "octet".

CoAP: Constrained Application Protocol [RFC7252]

CBOR: Concise Binary Object Representation [RFC7049]

CoRE: Constrained RESTful Environments, the field of work underlying [RFC6690], [RFC7049], [RFC7252], [RFC7641], [RFC7959], [RFC8075], and [RFC8323]

IoT: Internet of Things

JSON: JavaScript Object Notation [RFC8259]

The objective of the JSON and CBOR mappings defined in this document is to contain information of the formats specified in [RFC8288] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

2. Web Links in JSON and CBOR

2.1. Background

Web Linking [RFC8288] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery, for instance by using the CoAP protocol [RFC7252] and in conjunction with the CoRE resource directory [I-D.ietf-core-resource-directory].

2.2. Information Model

This section discusses the information model underlying the CORE Link Format payload.

An "application/link-format" document is a collection of Web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the collection of Web links to a JSON or CBOR array of links;
- o each link to a JSON object or CBOR map, mapping attribute names to attribute values.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "paramname"). The value can be a string, a language-tagged string, a boolean, or an array of these, as described below.

If the attribute value ("ptoken" or "quoted-string") is present, and a Link attribute with this name ("paramname") is present just once in the "link-value", the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the outer quotes removed and the backslash constructions evaluated) as defined in [RFC6690] and its referenced documents, before placing them in JSON strings (in the representation of which they may gain back additional decorations such as backslashes as defined in [RFC8259]).

Attribute values represented as per [RFC8187], e.g. for the "title*" attribute, are converted in a language-tagged string; the attribute name is then represented without the "*" character. A language-tagged string is represented as a CBOR map (JSON object) that carries the language tag as the key for a single member and the attribute value in UTF-8 form as its value.

If no attribute value ("ptoken" or "quoted-string") is present, the presence of the attribute name is indicated by using the Boolean value "true" as the value.

If a Link attribute ("paramname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values or "true"; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings or "true". (Note that [RFC6690] has cut down on the use of repeated parameter names; they are still allowed by [RFC8288] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel=

into JSON arrays.) Recipients MUST NOT accept documents that violate this requirement.

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value, with the latter converted to an IRI-Reference as per Section 3.2 of [RFC3987] (Rationale: The usage of "href" is consistent with the use of "href" as a query parameter for link-format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]. The usage of an IRI-Reference is consistent with the mandate in [RFC6690] that percent-encoding be processed. Note that the format is able to represent IRIs the URIs for which cannot be represented in [RFC6690] as not all percent-encoded constructions are amenable to the pre-processing required by [RFC6690].)

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 1 (informative).

```
links = [* link]
link = {
  href: tstr      ; resource URI
  * tstr => value
}
value1 = tstr      ; text value -- the normal case
        / { tstr => tstr } ; language tag and value
        / true       ; no value given, just the name
value = value1
        / [2* value1 ] ; repeats for two or more
```

Figure 1: CoRE Link Format Data Model (JSON)

2.3. Additional Encoding Step for CBOR

The above specification for JSON might have been used as is for the CBOR encoding as well. However, to further reduce message sizes, an extra encoding step is performed: "href" and some commonly occurring attribute names are encoded as small integers.

The substitution is defined in Table 1:

name	encoded value	origin
href	1	[RFC6690], [RFCthis]
rel	2	[RFC5988] Section 5.3
anchor	3	[RFC5988] Section 5.2
rev	4	[RFC5988] Section 5.3
hreflang	5	[RFC5988] Section 5.4
media	6	[RFC5988] Section 5.4
title	7	[RFC5988] Section 5.4
type	8	[RFC5988] Section 5.4
rt	9	[RFC6690] Section 3.1
if	10	[RFC6690] Section 3.2
sz	11	[RFC6690] Section 3.3
ct	12	[RFC7252] Section 7.2.1
obs	13	[RFC7641] Section 6

Table 1: Integer Encoding of common attribute names

This list of substitutions is fixed by the present specification; no future expansion of the list is foreseen. "href" as well as all attribute names in this list MUST be represented by their integer substitutions and MUST NOT use the attribute name in text form. Recipients MUST NOT accept documents that violate this requirement.

As a convenient reference, the resulting structure can be described in CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] as in Figure 2 (informative).

```

links = [* link]
link = {
  href => tstr      ; resource URI
  * label => value
}
href = 1
label = tstr / &(
  rel: 2,          anchor: 3,  rev: 4,
  hreflang: 5,     media: 6,   title: 7,
  type: 8,         rt: 9,      if: 10,
  sz: 11,         ct: 12,     obs: 13,
)
value1 = tstr      ; text value -- the normal case
           / { tstr => tstr } ; language tag and value
           / true     ; no value given, just the name
value = value1
           / [2* value1 ] ; repeats for two or more

```

Figure 2: CoRE Link Format Data Model (CBOR)

2.4. Converting JSON or CBOR to Link-Format

When a JSON or CBOR representation needs to be converted back to link-format, the above process is performed in inverse. Since link-format allows serializing link parameter values both in unquoted form ("ptoken") or in quoted form ("quoted-string"), a decision has to be made for each value. Where the syntax of "ptoken" does not allow the value to be represented, the quoted form clearly needs to be used. However, when both forms are possible, the decision is arbitrary. The recently republished Web Linking specification, [RFC8288], clarifies that this is indeed intended to be the case. However, previous specifications of link attributes, including those in [RFC5988] and [RFC6690], sometimes have made this decision in a specific way by only including one or the other alternative in the ABNF given for a link parameter. This requires a converter to know about all these cases, including those that have not been defined yet at the time of writing the converter. This problem becomes even harder by the fact that there is no central registry of link-attribute names.

Obviously, the conversion back to link-format needs to result in a valid link-format document. The reference implementation in Appendix A has addressed this problem with the following two rules:

- o Where a "ptoken" representation is possible, that is used instead of "quoted-string". This rule covers most of the special cases listed above.

- o As a special exception to the above rule, the four link attributes "anchor", "title", "rt", and "if" are always expressed as "quoted-string". This rule covers these specific four cases.

This set of rules is based on the hope that future definitions of link attributes will no longer hardcode one or the other serialization.

2.5. Examples

The examples in this section are based on an example on page 15 of [RFC6690] (Figure 3).

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 3: Example from page 15 of [RFC6690]

2.5.1. Link Format to JSON Example

The link-format document in Figure 3 becomes (321 bytes, line breaks shown are not part of the minimally-sized JSON document):

```
"[{\"href\":\"/sensors\",\"ct\":\"40\",\"title\":\"Sensor
Index\"},{\"href\":\"/sensors/temp\",\"rt\":\"temperature-
c\",\"if\":\"sensor\"},{\"href\":\"/sensors/light\",\"rt\":\"light-
lux\",\"if\":\"sensor\"},{\"href\":\"http://www.example.com/sensors/
t123\",\"anchor\":\"/sensors/
temp\",\"rel\":\"describedby\"},{\"href\":\"/t\",\"anchor\":\"/sensors/
temp\",\"rel\":\"alternate\"}] "
```

To demonstrate the handling of value-less and array-valued attributes, we extend the link-format example by examples of these (Figure 4; the "obs" attribute is defined in Section 6 of [RFC7641], while the "foo" attribute is for exposition only):

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor";obs,
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"
;rel="describedby";foo="bar";foo=3;ct=4711,
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 4: Example derived from page 15 of [RFC6690]

The link-format document in Figure 4 becomes the JSON document in Figure 5 (some spacing and indentation added):

```
[{"href":"/sensors","ct":"40","title":"Sensor Index"},
 {"href":"/sensors/temp","rt":"temperature-c","if":"sensor",
  "obs":true},
 {"href":"/sensors/light","rt":"light-lux","if":"sensor"},
 {"href":"http://www.example.com/sensors/t123",
  "anchor":"/sensors/temp","rel":"describedby",
  "foo":["bar","3"],"ct":"4711"},
 {"href":"/t","anchor":"/sensors/temp","rel":"alternate"}]
```

Figure 5: Example derived from page 15 of [RFC6690]

Note that the conversion is unable to convert the string-valued "ct" attribute to a number, which would be the natural type for a Content-Format value; similarly, both "foo" values are treated as strings independently of whether they are quoted or numeric in syntax.

2.5.2. Link Format to CBOR Example

This examples shows conversion from link format to CBOR format.

The link-format document in Figure 3 becomes (in CBOR diagnostic format):

```
[{1: "/sensors", 12: "40", 7: "Sensor Index"},
 {1: "/sensors/temp", 9: "temperature-c", 10: "sensor"},
 {1: "/sensors/light", 9: "light-lux", 10: "sensor"},
 {1: "http://www.example.com/sensors/t123", 3: "/sensors/temp",
  2: "describedby"},
 {1: "/t", 3: "/sensors/temp", 2: "alternate"}]
```

or, in hexadecimal (203 bytes):

```
85          # array(number of data items:5)
  a3        # map(# data item pairs:3)
    01      # unsigned integer(value:1,"href")
    68      # text string(8 bytes)
      2f73656e736f7273  # "/sensors"
    0c      # unsigned integer(value:12,"ct")
    62      # text(2)
      3430          # "40"
    07      # unsigned integer(value:7,"title")
    6c      # text string(12 bytes)
      53656e736f7220496e646578  # "Sensor Index"
  a3        # map(# data item pairs:3)
    01      # unsigned integer(value:1,"href")
```

```

6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
09      # unsigned integer(value:9,"rt")
6d      # text string(13 bytes)
74656d70657261747572
652d63  # "temperature-c"
0a      # unsigned integer(value:10,"if")
66      # text string(6 bytes)
73656e736f72  # "sensor"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
6e      # text string(14 bytes)
2f73656e736f72732f6c
69676874  # "/sensors/light"
09      # unsigned integer(value:9,"rt")
69      # text string(9 bytes)
6c696768742d6c7578  # "light-lux"
0a      # unsigned integer(value:10,"if")
66      # text string(6 bytes)
73656e736f72  # "sensor"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
78 23   # text string(35 bytes)
687474703a2f2f777777
2e6578616d706c652e63
6f6d2f73656e736f7273
2f74313233  # "http://www.example.com/sensors/t123"
03      # unsigned integer(value:3,"anchor")
6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
02      # unsigned integer(value:2,"rel")
6b      # text string(11 bytes)
6465736372696265646279  # "describedby"
a3      # map(# data item pairs:3)
01      # unsigned integer(value:1,"href")
62      # text string(2 bytes)
2f74     # "/t"
03      # unsigned integer(value:3,"anchor")
6d      # text string(13 bytes)
2f73656e736f72732f74
656d70  # "/sensors/temp"
02      # unsigned integer(value:2,"rel")
69      # text string(9 bytes)
616c7465726e617465  # "alternate"

```

Figure 6: Web Links Encoded in CBOR

3. IANA Considerations

3.1. Media types

This specification registers the following additional Internet Media Types:

Type name: application

Subtype name: link-format+json

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+json" media type are required to conform to the "application/json" Media Type and are therefore subject to the same encoding considerations specified in [RFC8259], Section 11.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in JSON.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): TEXT

Person & email address to contact for further information:
Carsten Bormann <cabo@tzi.org>

Intended usage: COMMON

Change controller: IESG

and

Type name: application

Subtype name: link-format+cbor

Required parameters: None

Optional parameters: None

Encoding considerations: Resources that use the "application/link-format+cbor" media type are required to conform to the "application/cbor" Media Type and are therefore subject to the same encoding considerations specified in [RFC7049], Section 7.

Security considerations: See Section 4 of [RFCthis].

Published specification: [RFCthis].

Applications that use this media type: Applications that interchange collections of Web links based on CoRE link format [RFC6690] in CBOR.

Additional information:

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): CBOR

Person & email address to contact for further information:
Kepeng Li <kepeng.lkp@alibaba-inc.com>

Intended usage: COMMON

Change controller: IESG

3.2. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the above media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. The ID for "application/link-format+cbor" is assigned from the "Expert Review" (0-255) range, while the ID for "application/link-format+json" is assigned from the "IETF review" range. The assigned IDs are show in Table 2.

Media type	Coding	ID	Reference
application/link-format+cbor	-	TBD64	[RFCthis]
application/link-format+json	-	TBD504	[RFCthis]

Table 2: CoAP Content-Format IDs

4. Security Considerations

The security considerations relevant to the data model of [RFC6690], as well as those of [RFC7049] and [RFC8259] apply.

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>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <<https://www.rfc-editor.org/info/rfc3987>>.
- [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>>.
- [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>>.
- [RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <<https://www.rfc-editor.org/info/rfc8187>>.
- [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>>.

- [RFC8288] Nottingham, M., "Web Linking", RFC 8288,
DOI 10.17487/RFC8288, October 2017,
<<https://www.rfc-editor.org/info/rfc8288>>.

5.2. Informative References

- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data
definition language (CDDL): a notational convention to
express CBOR data structures", draft-ietf-cbor-cddl-01
(work in progress), January 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-12 (work in progress), October 2017.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011,
<http://www.mnot.net/blog/2011/11/25/linking_in_json>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988,
DOI 10.17487/RFC5988, October 2010,
<<https://www.rfc-editor.org/info/rfc5988>>.
- [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>>.

- [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>>.
- [RUBY] "Information technology -- Programming languages -- Ruby", ISO/IEC 30170:2012, April 2012.

Appendix A. Reference implementation

A reference implementation of a converter from [RFC6690] link-format to JSON and CBOR (and back to link-format) in the programming language Ruby [RUBY] is reproduced below. (Note that this implementation does not handle [RFC8187]-encoded attributes.) For pretty-printing the binary CBOR, this uses the "cbor-diag" gem (Ruby library), which may need to be installed by "gem install cbor-diag".

```
# <CODE BEGINS>
require 'strscan'
require 'json'
require 'cbor-pretty'

class String
  def as_utf8
    force_encoding(Encoding::UTF_8)
  end
end

module CoRE
  module Links
    def self.map_to_true(a)
      Hash[a.map{ |t| [t, true]}]
    end

    PTOKENCHAR = %r"[\[\]\w!#-+\/<:-?^_`{~@]"
    QUOSTRCHAR = %r"{(?:[^\\"\\]|\\.)}" # to be used inside "
    ATTRCHAR = %r"[\w!#$&+.^'|~-]"
    MUSTBEQUOTED = map_to_true(%w{anchor title rt if})
    ANCHORNAME = "href"
    SCANATTR =
      %r{((#{ATTRCHAR}+)(?:=(?:({PTOKENCHAR}+)|"({QUOSTRCHAR}*)")?))} # "

    RAWMAPPINGS = <<-DATA
href: 1,  rel: 2,      anchor: 3,
rev: 4,   hreflang: 5, media: 6,
title: 7, type: 8,     rt: 9,
if: 10,   sz: 11,      ct: 12,
```

```

obs: 13,
  DATA

  MAPPINGS = Hash.new {|h, k| k}

  RAWMAPPINGS.scan(/([-\w]+\s*:\s*([-\w]+),)/) do |n, v|
    MAPPINGS[n] = Integer(v)
  end

  def self.parse(*args)
    WLNK.parse(*args)
  end

  class WLNK
    attr_accessor :resources
    def initialize(r = []) # make sure the keys are strings
      @resources = r.to_ary # make sure it's an Array
    end
    def self.parse(s, robust = true)
      wl = WLNK.new
      ss = StringScanner.new(s.as_utf8)
      ss.skip(/\s+/) if robust
      while ss.scan(%r{<([>]+)>})
        res = { ANCHORNAME => ss[1].as_utf8 }
        ss.skip(/\s*/) if robust
        while ss.skip(//)
          ss.skip(/\s*/) if robust
          unless ss.scan(SCANATTR)
            raise ArgumentError, "must have attribute behind ';'
              at: #{ss.peek(20).inspect} (byte #{ss.pos})"
          end
          key = ss[1].as_utf8
          value = ss[2] ||
            (ss[3] ? ss[3].gsub(/\\(.)/) { $1 } : true)
          if res[key]
            res[key] = Array(res[key]) << value
          else
            res[key] = value
          end
          ss.skip(/\s*/) if robust
        end
        wl.resources << res
        break unless ss.skip(//)
        ss.skip(/\s*/) if robust
      end
      ss.skip(/\s*/) if robust
      raise ArgumentError, "link-format unparseable at:
        #{ss.peek(20).inspect} (byte #{ss.pos})" unless ss.eos?
    end
  end

```

```

        wl
    end
    def to_json
        JSON.pretty_generate(@resources)
    end
    def to_cbor
        CBOR.encode(@resources.map { |r|
            Hash[r.map { |k, v| [MAPPINGS[k], v] }]])
    end
    def to_wlnk
        resources.map do |res|
            res = res.dup
            u = res.delete(ANCHORNAME)
            ["<#{u}>", *res.map { |k, v| wlnk_item(k, v) }].join(';')
        end.join(",")
    end
    private
    def wlnk_item(k, v)
        case v
        when String
            if MUSTBEQUOTED[k] || v !~ /\A#{PTOKENCHAR}+\z/
                "#{k}=\"#{v.gsub(/[\\"]/) { |x| "\\#{x}" }}\""
            else
                "#{k}=#{v}"
            end
        when Array
            v.map { |v1| wlnk_item(k, v1) }.join(';')
        when true
            "#{k}"
        else
            fail "Don't know how to represent #{k=>v}.inspect"
        end
    end
end
end
end
end

lf = CoRE::Links.parse(ARGF.read)

puts lf.to_json           # JSON
puts CBOR.pretty(lf.to_cbor) # CBOR "pretty" binary form
puts lf.to_wlnk          # RFC 6690 link-format
# <CODE ENDS>

```

Acknowledgements

Special thanks to Bert Greevenbosch who was an author on the initial version of a contributing document as well as the original author on the CDDL notation.

Hannes Tschofenig made many helpful suggestions for improving this document.

Authors' Addresses

Kepeng LI
Alibaba Group
Wenyixi Road, Yuhang District
Hangzhou, Zhejiang 311121
China

Email: kepeng.lkp@alibaba-inc.com

Akbar Rahman
InterDigital Communications, LLC
1000 Sherbrooke Street West
Montreal, Quebec H3A 3G4
Canada

Phone: +1-514-585-0761
Email: akbar.rahman@interdigital.com

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

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

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

T. Fossati
Nokia
K. Hartke
Ericsson
C. Bormann
Universitaet Bremen TZI
July 02, 2018

Multipart Content-Format for CoAP
draft-ietf-core-multipart-ct-01

Abstract

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of several different media types into a single CoAP message-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 January 3, 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. Multipart Content-Format Encoding	3
3. Usage Examples	3
3.1. Observing Resources	3
3.2. Implementation hints	4
4. IANA Considerations	5
4.1. Registration of media type application/multipart-core . .	5
4.2. Registration of a Content-Format identifier for application/multipart-core	6
5. Security Considerations	6
6. References	7
6.1. Normative References	7
6.2. Informative References	7
Acknowledgements	7
Authors' Addresses	8

1. Introduction

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of several different media types into a single CoAP [RFC7252] message-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.

Applications using the application/multipart-core Content-Format define the internal structure of the application/multipart-core representation.

For example, one way to structure the sub-types specific to an application/multipart-core container is to always include them at the same fixed position. This specification allows to indicate that an optional part is not present by substituting a null value for the representation of the part.

Optionally, an application might use the general format defined here, but also register a new media type and an associated Content-Format identifier -- typically one in the range 10000-64999 -- instead of using application/multipart-core.

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. (Future extensions might want to include additional alternative ways of specifying the media type of a representation in such a position.)

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

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.

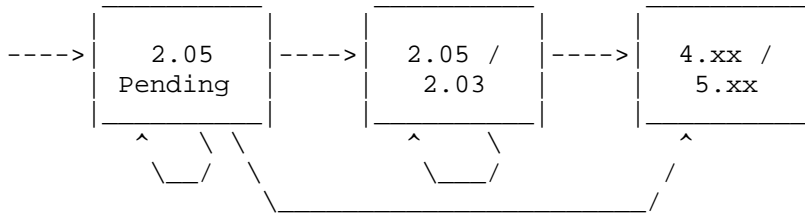


Figure 2: Sequence of Notifications:

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.

3.2. 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..66535

Table 1: Serialization of content-format

Serialization	Length
0x40..0x57	0..23
0x58 0xnn	24..255
0x59 0xnn 0xnn	256..66535
0x5a 0xnn 0xnn 0xnn 0xnn	66536..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 about its content-format (here: 0x82 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.

4. IANA Considerations

4.1. 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 potentially several media types into one, e.g., EST-CoAP [I-D.ietf-ace-coap-est]

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: CoRE WG

Change controller: IESG

Provisional registration? (standards tree only): no

4.2. 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 IETF Review space (specifically, 256..999):

Media Type	Encoding	ID	Reference
application/multipart-core	--	TBD1	RFCthis

5. 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.

6. References

6.1. Normative References

- [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>>.

6.2. Informative References

- [I-D.ietf-ace-coap-est] Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuhed, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-03 (work in progress), June 2018.
- [I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.
- [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>>.

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
Nokia

Email: thomas.fossati@nokia.com

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

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

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

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2018

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
RISE SICS
June 27, 2018

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-13

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.

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 December 29, 2018.

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	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	21
4.3. Signaling Messages	23
5. The COSE Object	23
5.1. Kid Context	24
5.2. Nonce	25
5.3. Plaintext	26
5.4. Additional Authenticated Data	27
6. OSCORE Header Compression	28
6.1. Encoding of the OSCORE Option Value	28
6.2. Encoding of the OSCORE Payload	30
6.3. Examples of Compressed COSE Objects	30
7. Message Binding, Sequence Numbers, Freshness and Replay Protection	32
7.1. Message Binding	32
7.2. Sequence Numbers	32
7.3. Freshness	33
7.4. Replay Protection	33
7.5. Losing Part of the Context State	34
8. Processing	35
8.1. Protecting the Request	36
8.2. Verifying the Request	36
8.3. Protecting the Response	37
8.4. Verifying the Response	38
9. Web Linking	40
10. CoAP-to-CoAP Forwarding Proxy	41
11. HTTP Operations	42
11.1. The HTTP OSCORE Header Field	42
11.2. CoAP-to-HTTP Mapping	43
11.3. HTTP-to-CoAP Mapping	43
11.4. HTTP Endpoints	44

11.5. Example: HTTP Client and CoAP Server	44
11.6. Example: CoAP Client and HTTP Server	46
12. Security Considerations	47
12.1. End-to-end Protection	47
12.2. Security Context Establishment	48
12.3. Master Secret	48
12.4. Replay Protection	48
12.5. Client Aliveness	49
12.6. Cryptographic Considerations	49
12.7. Message Segmentation	49
12.8. Privacy Considerations	50
13. IANA Considerations	50
13.1. COSE Header Parameters Registry	51
13.2. CoAP Option Numbers Registry	51
13.3. CoAP Signaling Option Numbers Registry	51
13.4. Header Field Registrations	51
13.5. Media Type Registrations	52
13.6. CoAP Content-Formats Registry	54
14. References	54
14.1. Normative References	54
14.2. Informative References	56
Appendix A. Scenario Examples	58
A.1. Secure Access to Sensor	58
A.2. Secure Subscribe to Sensor	59
Appendix B. Deployment Examples	60
B.1. Master Secret Used Once	60
B.2. Master Secret Used Multiple Times	61
Appendix C. Test Vectors	62
C.1. Test Vector 1: Key Derivation with Master Salt	62
C.2. Test Vector 2: Key Derivation without Master Salt	63
C.3. Test Vector 3: Key Derivation with ID Context	64
C.4. Test Vector 4: OSCORE Request, Client	66
C.5. Test Vector 5: OSCORE Request, Client	67
C.6. Test Vector 6: OSCORE Request, Client	68
C.7. Test Vector 7: OSCORE Response, Server	69
C.8. Test Vector 8: OSCORE Response with Partial IV, Server	70
Appendix D. Overview of Security Properties	71
D.1. Supporting Proxy Operations	71
D.2. Protected Message Fields	72
D.3. Uniqueness of (key, nonce)	73
D.4. Unprotected Message Fields	74
Appendix E. CDDL Summary	76
Acknowledgments	77
Authors' Addresses	77

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 [RFC5246] 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 Observe [RFC7641], Block-wise [RFC7959], No-Response [RFC7967], and PATCH and FETCH [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).

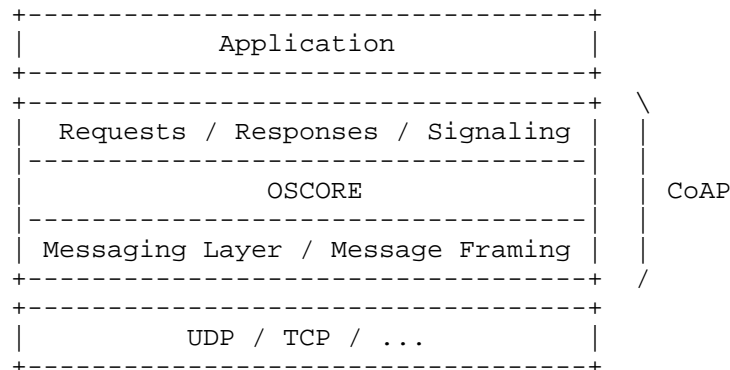


Figure 1: Abstract Layering of CoAP with OSCORE

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 does not depend on underlying layers, and can be used with non-IP transports (e.g., [I-D.bormann-6lo-coap-802-15-4e]). 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.

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.

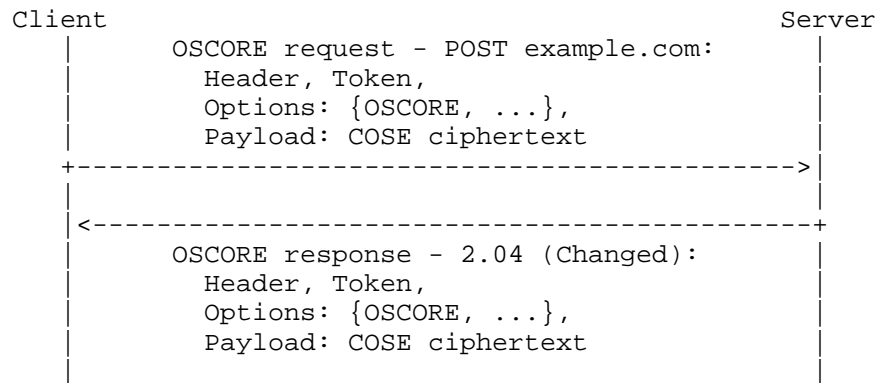


Figure 2: Sketch of CoAP with OSCORE

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 (see Figure 3, which extends Table 4 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 (KDF).

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.

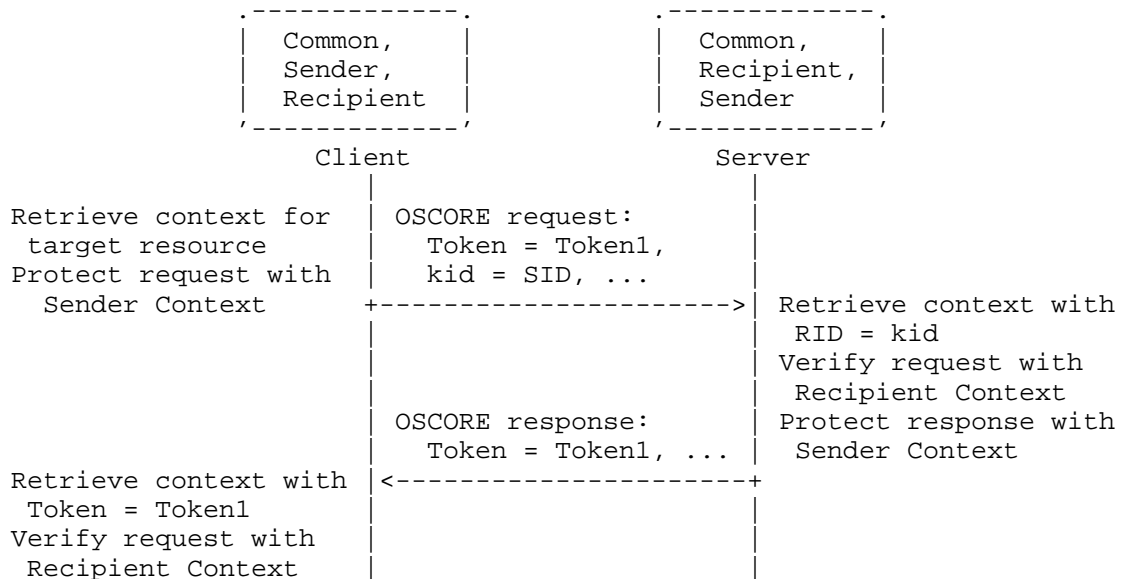


Figure 4: Retrieval and Use of the Security Context

The Common Context contains the following parameters:

- o AEAD Algorithm. The COSE AEAD algorithm to use for encryption.
- o Key Derivation Function. The HMAC based 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 traffic keys and IVs.
- o Master Salt. Optional variable length byte string containing the salt used to derive traffic keys and IVs.
- o ID Context. Optional variable length byte string providing additional information to identify the Common Context and to derive traffic keys and IVs.
- o Common IV. Byte string derived from Master Secret, Master Salt, and ID Context. Length is determined by the AEAD Algorithm.

The Sender Context contains the following parameters:

- o Sender ID. Byte string used to identify the Sender Context, to derive traffic keys and IVs, and to assure unique nonces. Maximum length is determined by the AEAD Algorithm.
- o Sender Key. Byte string containing the symmetric 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 protect requests and certain responses, e.g. Observe notifications. Used as 'Partial IV' [RFC8152] to generate unique nonces for the AEAD. Maximum value is determined by the AEAD Algorithm.

The Recipient Context contains the following parameters:

- o Recipient ID. Byte string used to identify the Recipient Context, to derive traffic keys and IVs, and to assure unique nonces. Maximum length is determined by the AEAD Algorithm.
- o Recipient Key. Byte string containing the symmetric 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.

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 keys to be used.

3.2. Establishment of Security Context Parameters

The parameters in the security context are derived 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 Key Derivation Function (KDF)
 - * Default is HKDF SHA-256
- o Replay Window Type and Size
 - * 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 KDF 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:

```
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 keys and the empty byte string when deriving the Common IV. The encoding is described in Section 5.
- 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/IV 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.

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 in [RFC5869] 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. The requirement that Sender ID SHALL be unique in the set of all security contexts using the same Master Secret, Master Salt, and ID Context guarantees unique (key, nonce) pairs, which avoids nonce reuse.

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. 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 generated by the client (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.

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

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.4).

4.1.1. 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.

4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations, see Appendix D.1.

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 OSCORE error responses at OSCORE-unaware intermediary nodes. A server MAY set a Class U Max-Age message field with value zero to OSCORE error responses, which are described in Sections 7.4, 8.2, and 8.4. Such a message field is then processed according to Section 4.1.2.

Successful OSCORE responses do not need to include an Outer Max-Age option since the responses are non-cacheable by construction (see Section 4.2).

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 SHOULD 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 in responses (with or without the Observe option) to Observe registrations.

[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). The Partial IV of the registration is also used as 'request_piv' of associated cancellations (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). 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.

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.

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).

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

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 key lengths, IV length, 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

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.
 - * 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) containing an ID Context (see Section 3.1). This parameter MAY be present in requests and MUST NOT be present in responses. 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. 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 In case of group communication [I-D.ietf-core-oscore-groupcomm], a group identifier can be used as ID Context to enable different security contexts for a server belonging to multiple groups.

The Sender ID and Context ID 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, 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

5.2. Nonce

The AEAD nonce is constructed in the following way (see Figure 8):

1. left-padding the Partial IV (PIV) in network byte order with zeroes to exactly 5 bytes,
2. left-padding the Sender ID of the endpoint that generated the Partial IV (ID_PIV) in network byte order with zeroes to exactly nonce length minus 6 bytes,
3. concatenating the size of the ID_PIV (a single byte S) with the padded ID_PIV and the padded PIV,
4. and then XORing with the Common IV.

Note that in this specification only 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.3.

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.3. 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 Section 7.5.2. For processing instructions see Section 8.

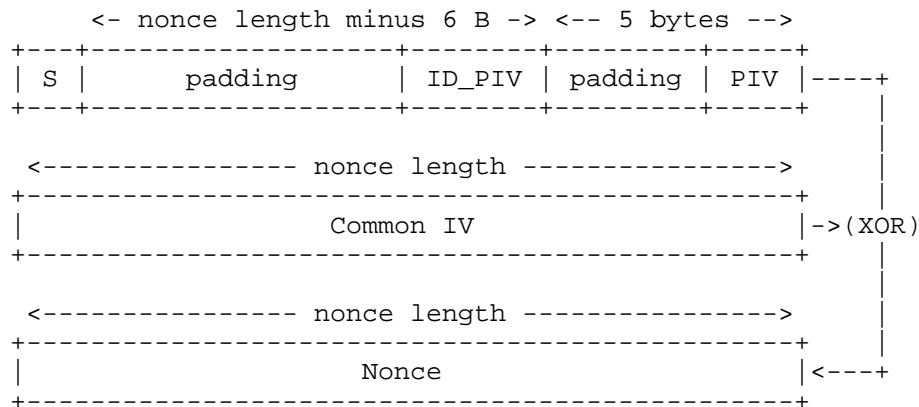


Figure 8: AEAD Nonce Formation

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).

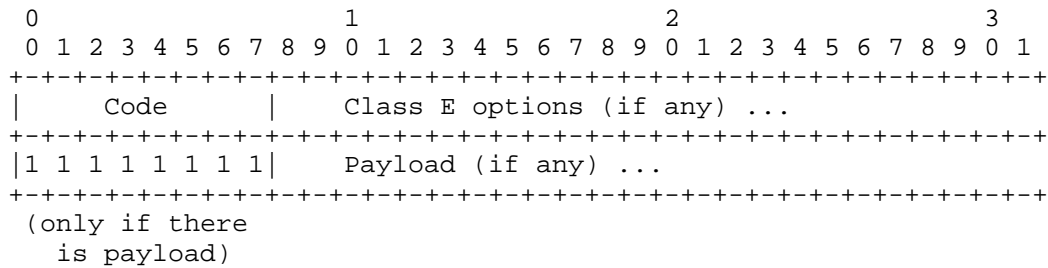


Figure 9: Plaintext

NOTE: The plaintext contains all CoAP data that needs to be encrypted end-to-end between the endpoints.

5.4. Additional Authenticated Data

The `external_aad` SHALL be a CBOR array as defined below:

```

external_aad = [
  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), with one exception: in case of protection or verification of Observe cancellations, the `request_piv` contains the value of the 'Partial IV' in the COSE object of the corresponding registration (see Section 4.1.3.5.1).

- 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_add` as described in Section 5.3 of [RFC8152].

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:

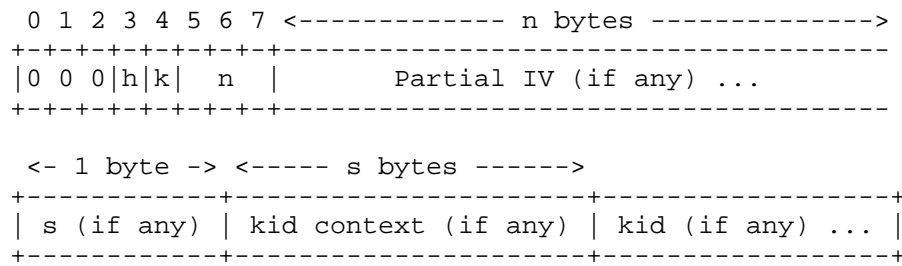


Figure 10: The OSCORE Option Value

- o The first byte of flag bits encodes the following set of flags 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 3 of Section 8.2.
- 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

Option Value: 09 05 25 (3 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (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

Option Value: 09 00 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (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

Option Value: 19 05 05 44 61 6c 65 6b (8 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (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

Option Value: (0 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (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

Option Value: 01 07 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (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.3, and in particular depends on a correct usage of Partial IVs. 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 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 receiving a notification 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 key, or from accepting replayed messages, an endpoint needs to handle the situation of losing rapidly changing parts of the context, such as the request Token, Sender Sequence Number, Replay Window, and Notification Numbers. These are typically stored in RAM and therefore lost in the case of an unplanned reboot.

After boot, an endpoint can either use a persistently stored complete or partial security context, or establish a new security context with each endpoint it communicates with. However, establishing a fresh security context may have a non-negligible cost in terms of, e.g., power consumption.

If the endpoint uses a persistently stored partial security context, it MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously received messages. Some ways to achieve this are described in the following sections.

7.5.1. Sequence Number

To prevent reuse of Sender Sequence Numbers, 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 in persistent memory. After boot, the endpoint initiates the Sender Sequence Number to the value stored in persistent memory + K . Storing to persistent memory can be costly. The value K gives a trade-off between the number of storage operations and efficient use of Sender Sequence Numbers.

7.5.2. Replay Window

To prevent accepting replay of previously received requests, the server may perform the following procedure after boot:

- o For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with the Echo option [I-D.ietf-core-echo-request-tag] to get a request with verifiable freshness. The server MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response.

If the server using the Echo option can verify a second request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window.

7.5.3. Replay of 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.

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.

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 SHOULD 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 SHOULD contain the string "Security context not found".
3. Verify the 'Partial IV' parameter 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 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 request was a registration, 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 nonce from the request is used.
 - * If the Partial IV is present in the response, compute the 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 either the client has previously received a successful notification to the

registration (active observation) or Inner Observe is present, 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.4.

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 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. It is recommended to use a key establishment protocol which provides forward secrecy whenever possible. 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.

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 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*.

In order to prevent cryptanalysis when the same plaintext is repeatedly encrypted by many different users with distinct keys, the 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).

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.4. 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 "TBDx" 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]]

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 (any)	TBD1	OSCORE	[[this document]]

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]]

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

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-Format" 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]]

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

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-12 (work in progress), May 2018.
- [I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-01 (work in progress), March 2018.
- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-01 (work in progress), March 2018.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-01 (work in progress), March 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-05 (work in progress), March
2018.
- [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.
- [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.

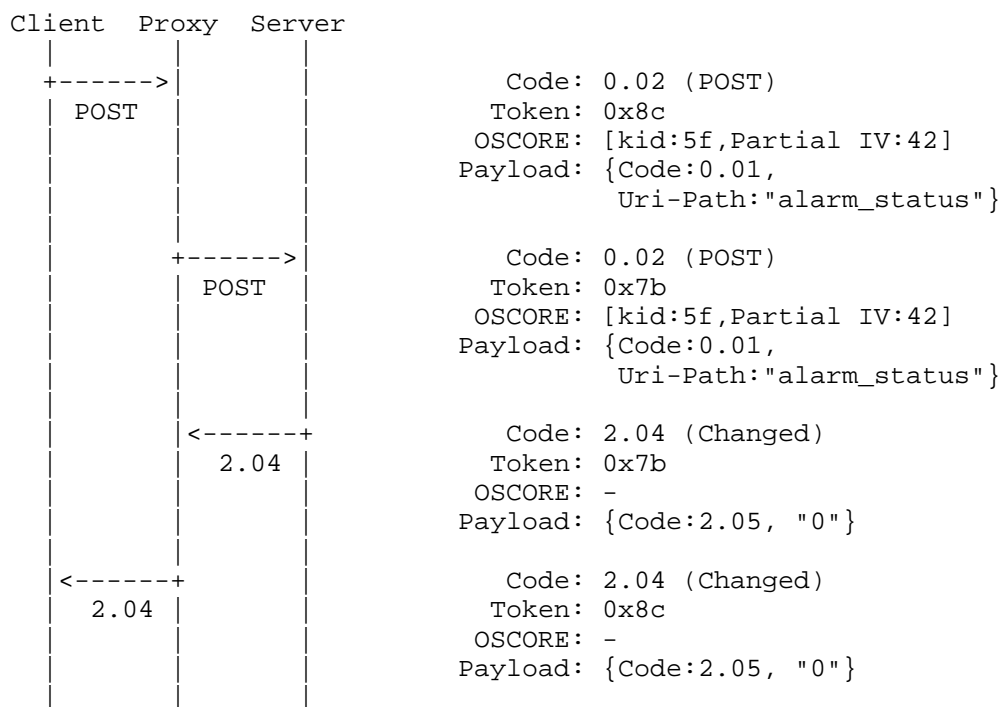


Figure 12: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

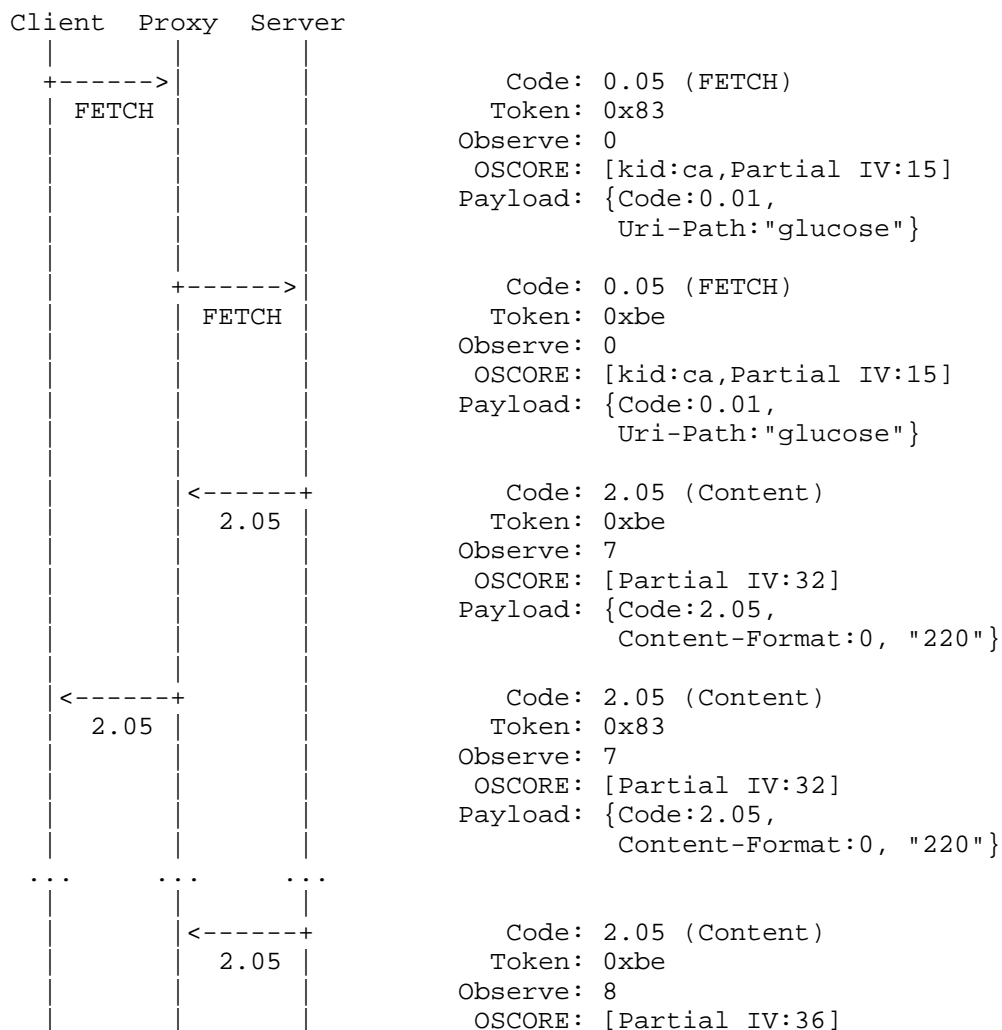
The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("0") are encrypted.

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.



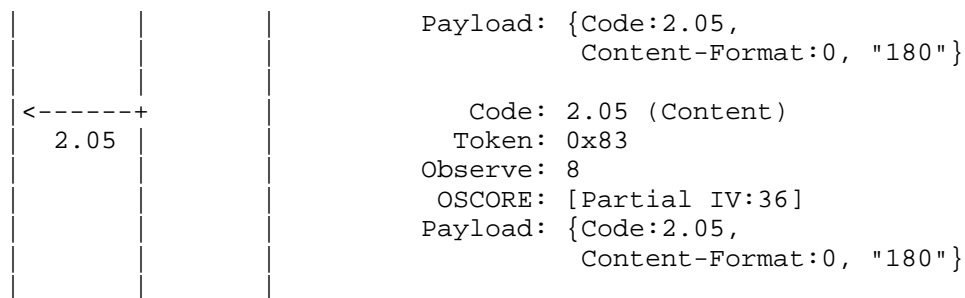


Figure 13: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

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

Two examples complying with the requirements on the security context parameters (Section 3.3) are given in this section.

B.1. Master Secret Used Once

An application may derive a security context once and use it for the lifetime of a device. For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device. This specification describes techniques for persistent storage of the security context and synchronization of sequence numbers (see Section 7.5) to ensure that security is maintained with the existing security context.

B.2. Master Secret Used Multiple Times

Section 12.2 recommends the use of a key establishment protocol providing forward secrecy of the Master Secret.

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 an application allowing new security contexts to be derived from input parameters pre-established between client and server for this purpose: in particular Master Secret, Master Salt and Sender/Recipient ID (see Section 3.2):

- o The client generates an ID Context which has previously not been used with the pre-established input parameters and derives a new security context. ID context may be pseudo-random and large for stochastic uniqueness, but care must be taken e.g. to avoid re-use of the same seed for random number generation. Using this new security context, the client generates an OSCORE request with (kid context, kid) = (ID Context, Sender ID) in the OSCORE option.
- o The server receiving such an OSCORE request with kid matching the Recipient ID of pre-established input parameters, but with a new kid context, derives the security context using ID Context = kid context. If the message verifies then a new security context with this ID Context is stored in the server, and used in the response. Further requests with the same (kid context, kid) are verified with this security context.

As an alternative procedure to reduce the subsequent overhead in requests due to kid context, the verification of a message with a new ID Context may trigger the server to generate a new kid to replace the Client Sender ID in future requests. A client may e.g. indicate support for such a procedure by requesting a special well-known URI and receive the new kid in the response, which together with the input parameters and the ID context is used to derive the new security context which may be identified only by its kid. The details are out of scope for this specification.

The procedures may be complemented with the use of the Echo option for verifying the aliveness of the client requesting a new security context.

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.

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 KDF.

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)

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)

C.2. Test Vector 2: Key Derivation without Master Salt

In this test vector, the default values are used for AEAD Algorithm, KDF, 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)

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)

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 KDF.

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)

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)

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: 0x4622d4dd6d944168eeffb54987c (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: 0x4622d4dd6d944168eeffb549868 (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: 0x321b26943253c7ffb6003b0b64d74041 (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: 0x321b26943253c7ffb6003b0b64d74041 (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 carrying the ID Context in the message, using 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): 0x44022f8eef9bbf7a396c6f63616c686f73746b19140837cbf3210017a2d3ff4ed339a5a379b0b8bc731fffb0 (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. 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]. The client and server are assumed to be honest, but proxies and gateways are only trusted to perform their intended operations. Forwarding is specified in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. HTTP-CoAP translation is specified in [RFC8075]. Intermediaries translating between different transport layers are intended to perform just that.

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.2. 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.3). 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.3. 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 (subset of cases with 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.4. Unprotected Message Fields

This section lists and discusses issues with unprotected message fields.

D.4.1. CoAP Header Fields

- o Version. The CoAP version [RFC7252] is not expected to be sensitive to disclose. 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]. 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. CoAP proxies are allowed to change Token and Token Length between UDP hops. However, modifications of Token and Token Length during a UDP hop 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. The use of FETCH reveals no more than what is revealed by the Outer Observe option. Changing the Outer Code may be a denial-of-service attack by causing errors in the proxy processing.
- 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 the other endpoint received the previous message. By sending a RST, an attacker may be able to cancel an observation, make one endpoint believe the other endpoint is alive, or make one endpoint believe that the other endpoint is missing some context. By changing a NON to a CON, the

attacker can cause the receiving endpoint to respond to messages for which no response 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.4.2. 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 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 since they are allowed to be changed by a forward proxy. 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 an OSCORE-unaware 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 size and distributions of notifications over time may reveal information about the content or nature of the notifications.
- 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.4.3. 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.

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.4.4. 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 [RFC5246] 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, Esko Dijk, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Michael Richardson, Jim Schaad, Peter van der Stok, Dave Thaler, 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

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
RISE SICS

Email: ludwig.seitz@ri.se

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2018

M. Tiloca
RISE SICS
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
June 28, 2018

Secure group communication for CoAP
draft-ietf-core-oscore-groupcomm-02

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 is defined how OSCORE should be used in a group communication setting, while fulfilling the same security requirements for request messages and related response messages. Source authentication of all messages exchanged within the group is ensured, by means of digital signatures produced through private keys of sender endpoints 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 December 30, 2018.

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. OSCORE Security Context	5
2.1. Management of Group Keying Material	7
3. The COSE Object	7
3.1. Example: Request	9
3.2. Example: Response	10
4. Message Processing	10
4.1. Protecting the Request	10
4.2. Verifying the Request	11
4.3. Protecting the Response	11
4.4. Verifying the Response	11
5. Synchronization of Sequence Numbers	12
6. Responsibilities of the Group Manager	12
7. Security Considerations	13
7.1. Group-level Security	14
7.2. Uniqueness of (key, nonce)	14
7.3. Collision of Group Identifiers	14
8. IANA Considerations	15
9. Acknowledgments	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Appendix A. Assumptions and Security Objectives	18
A.1. Assumptions	18
A.2. Security Objectives	20
Appendix B. List of Use Cases	21
Appendix C. Example of Group Identifier Format	23
Appendix D. Set-up of New Endpoints	24
D.1. Join Process	24
D.2. Provisioning and Retrieval of Public Keys	27
D.3. Group Joining Based on the ACE Framework	29
Appendix E. Examples of Synchronization Approaches	29
E.1. Best-Effort Synchronization	29
E.2. Baseline Synchronization	30
E.3. Challenge-Response Synchronization	30

Appendix F. No Verification of Signatures	32
Appendix G. Document Updates	32
G.1. Version -01 to -02	32
G.2. Version -00 to -01	33
Authors' Addresses	34

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 and improve performance. 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, and replay protection between a sending endpoint and a receiving endpoint possibly involving intermediary endpoints. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which finally replaces the authenticated and encrypted fields in the protected message.

This document describes group OSCORE, providing end-to-end security of CoAP messages exchanged between members of a group. In particular, the described approach defines how OSCORE should be used in a group communication setting, so that end-to-end security is assured by using the same security method. That is, end-to-end security is assured for (multicast) CoAP requests sent by client endpoints to the group and for related CoAP responses sent as reply by multiple server endpoints. 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 to protect hop-by-hop communication between a sender endpoint and a proxy (and vice versa), and between a proxy and a recipient endpoint (and vice versa).

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 parameters (Common Context of the group's Security Context, see Section 2). That is, the term group used in this specification refers to a "security group", not to be confused with network/multicast groups or application groups.
- o Group Manager (GM): entity responsible for a set of OSCORE groups. Each endpoint in a group securely communicates with the respective GM, which is not required to be an actual group member and to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 6.
- o Silent server: member of a group that never replies back after receiving request messages.
- o Group ID: group identifier assigned to the group. Group IDs are unique within the set of groups of a same Group Manager.
- o Endpoint ID: Sender ID of the endpoint, as defined in [I-D.ietf-core-object-security]. An Endpoint ID is provided to an endpoint upon joining a group, is valid only within that group,

and is unique within the same group. Endpoints which are configured only as silent servers do not have an Endpoint ID.

- o Group request: CoAP request message sent by a client endpoint in the group to all server endpoints in that group.
- o Source authentication: evidence that a received message in the group originated from a specifically identified group member. This also provides assurances that the message was not tampered with by a different group member or by a non-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]. Each endpoint in a group stores:

1. one Common Context, shared by all the endpoints in the group. In particular:
 - * All the endpoints in the group agree on the same COSE AEAD algorithm.
 - * The ID Context parameter stores the Group ID of the group, which is used to retrieve the Security Context for processing messages intended to the group's endpoints (see Section 4). The choice of the Group ID for a given group's Security Context is application specific. An example of specific formatting of the Group ID that would follow this specification is given in Appendix C. It is the role of the application to specify how to handle possible collisions.
 - * A new parameter Counter Signature Algorithm is included, and its value identifies the algorithm used for source authenticating messages sent within the group, by means of a counter signature (see Section 4.5 of [RFC8152]). Its value is immutable once the Common Context is established. All the endpoints in the group agree on the same counter signature algorithm. The list of supported signature algorithms is part of the group communication policy and MUST include the EdDSA signature algorithm ed25519 [RFC8032].
2. one Sender Context, unless the endpoint is configured exclusively as silent server. The Sender Context is used to secure outgoing group messages and is initialized according to Section 3 of [I-D.ietf-core-object-security], once the endpoint has joined the group. In practice, the symmetric keying material in the Sender

Context of the sender endpoint is shared with all the recipient endpoints that have received group messages from that same sender endpoint. Besides, in addition to what is defined in [I-D.ietf-core-object-security], the Sender Context stores also the endpoint's public-private key pair.

3. one Recipient Context for each distinct endpoint from which group messages are received, used to process such incoming messages. The recipient endpoint creates a new Recipient Context upon receiving an incoming message from another endpoint in the group for the first time (see Section 4.2 and Section 4.4). In practice, the symmetric keying material in a given Recipient Context of the recipient endpoint is shared with the associated sender endpoint from which group 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 group 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
Sender Context	Endpoint's own private key
Sender Context	Endpoint's own public 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 endpoint relies on the sender endpoint's public key, in order to verify the counter signature conveyed in the COSE Object.

If not already stored in the Recipient Context associated to the sender endpoint, the recipient endpoint retrieves the public key from a trusted key repository. In such a case, the correct binding between the sender endpoint and the retrieved public key must be

assured, for instance by means of public key certificates. Further discussion about how public keys can be handled and retrieved in the group is provided in Appendix D.2.

The Sender Key/IV stored in the Sender Context and the Recipient Keys/IVs stored in the Recipient Contexts are derived according to the same scheme defined in Section 3.2 of [I-D.ietf-core-object-security].

2.1. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, the adoption of key management schemes for secure revocation and renewal of Security Contexts and group keying material should be considered.

Consistently with the security assumptions in Appendix A.1, it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value 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 in order to preserve backward security and forward security in the group.

In particular, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter must also be distributed. An example of Group Identifier 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 Group Identifier.

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.

3. The COSE Object

When creating a protected CoAP message, an endpoint in the group computes the COSE object using the untagged COSE_Encrypt0 structure [RFC8152] as defined in Section 5 of [I-D.ietf-core-object-security], with the following modifications.

- o The value of the 'kid' parameter in the 'unprotected' field of response messages SHALL be set to the Endpoint ID of the endpoint transmitting the message, i.e. the Sender ID.

- o The 'unprotected' field SHALL additionally include the following parameter:
 - * CounterSignature0 : its value is set to the counter signature of the COSE object, computed by the endpoint by means of its own private key as described in Section 4.5 of [RFC8152]. The presence of this parameter is explicitly signaled, by using the reserved sixth least significant bit of the first byte of flag bits in the value of the OSCORE Option (see Section 6.1 of [I-D.ietf-core-object-security]).
- o The Additional Authenticated Data (AAD) considered to compute the COSE object is extended with the counter signature algorithm used to protect group messages. In particular, with reference to Section 5.4 of [I-D.ietf-core-object-security], the 'algorithms' array in the external_aad SHALL also include 'alg_countersign', which contains the Counter Signature Algorithm from the Common Context (see Section 2).

```
external_aad = [  
  ...  
  algorithms : [alg_aead : int / tstr , alg_countersign : int / tstr],  
  ...  
]
```

- o 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.
 - * The fourth least significant bit of the first byte of flag bits SHALL be set to 1, to indicate the presence of the 'kid' parameter for both group requests and responses.
 - * The fifth least significant bit of the first byte of flag bits MUST be set to 1 for group requests, to indicate the presence of the 'kid context' parameter in the OSCORE payload. The kid context flag MAY be set to 1 for responses.
 - * The sixth least significant bit of the first byte of flag bits is originally marked as reserved in [I-D.ietf-core-object-security] and its usage is defined in this specification. This bit is set to 1 if the 'CounterSignature0' parameter is present, or to 0 otherwise. In order to ensure source authentication of group messages as described in this specification, this bit SHALL be set to 1.
 - * The 'kid context' value encodes the Group Identifier value (Gid) of the group's Security Context.

- * The following q bytes (q given by the Counter Signature Algorithm specified in the Security Context) encode the value of the 'CounterSignature0' parameter including the counter signature of the COSE object.
- * 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.

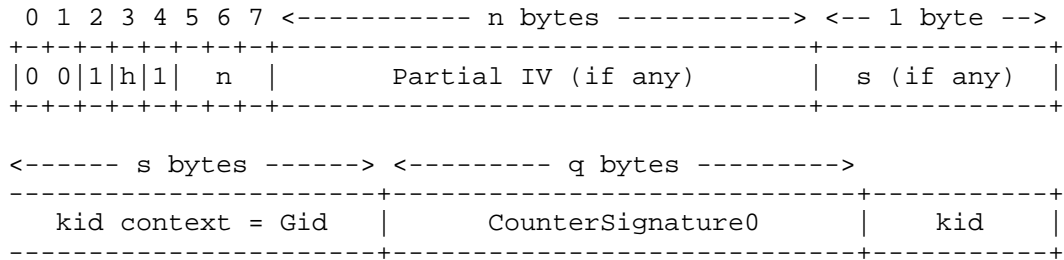


Figure 2: OSCORE Option Value

3.1. Example: Request

Request with kid = 0x25, Partial IV = 5 and kid context = 0x44616c, assuming 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 in this example. The ciphertext in this example is 14 bytes long.

Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 9:COUNTERSIGN },
  h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (85 bytes):

Flag byte: 0b00111001 = 0x39

Option Value: 39 05 03 44 61 6c COUNTERSIGN 25 (7 bytes + size of COUNTERSIGN)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

3.2. Example: Response

Response with kid = 0x52. COUNTERSIGN is the CounterSignature0 byte string as described in Section 3 and is 64 bytes long in this example. The ciphertext in this example is 14 bytes long.

Before compression (88 bytes):

```
[  
  h'',  
  { 4:h'52', 9:COUNTERSIGN },  
  h'60b035059d9ef5667c5a0710823b'  
]
```

After compression (80 bytes):

Flag byte: 0b00101000 = 0x28

Option Value: 28 COUNTERSIGN 52 (2 bytes + size of COUNTERSIGN)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b (14 bytes)

4. 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, and message ordering.

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 receiver endpoint MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 3, without sending 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.

4.1. 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 encoding of the compressed COSE object is modified as described in Section 3.

4.2. 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 is modified as described in Section 3. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Group ID ('kid context'), then the server creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes 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.

4.3. 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 encoding of the compressed COSE object is modified as described in Section 3.

4.4. 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. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Group

ID ('kid context'), then the client creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes 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.

5. Synchronization of Sequence Numbers

Upon joining the group, new servers are not aware of the 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. The same holds when a server loses synchronization with sequence numbers of clients, for instance after a device reboot.

The exact way to address this issue depends on the specific use case and its synchronization requirements. The list of methods to handle synchronization of 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.

6. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

- o Creating and managing OSCORE groups. This includes the assignment of a Group ID to every newly created group, as well as ensuring uniqueness of Group IDs within the set of its OSCORE groups.
- o Defining policies for authorizing the joining of its OSCORE groups. Such policies can be enforced by a third party, which is in a trust relation with the Group Manager and enforces join policies on behalf of the Group Manager.
- o Driving the join process to add new endpoints as group members.
- o Establishing Security Common Contexts and providing them to authorized group members during the join process, together with a corresponding Security Sender Context.

- o Generating and managing Endpoint IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process. This includes ensuring uniqueness of Endpoints IDs within each of its OSCORE groups.
- o Defining a set of supported signature algorithms as part of the communication policy of each of its OSCORE groups, and signalling it to new endpoints during the join process.
- o Defining the methods to handle loss of synchronization with sequence numbers as part of the communication policy of each of its OSCORE groups, and signaling the one(s) to use to new endpoints during the join process.
- o Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).
- o Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistently with the key management scheme used in the group joined by the new endpoint.
- o Updating the Group ID of its OSCORE groups, upon renewing the respective Security Context.

The Group Manager may additionally be responsible for the following tasks:

- o Acting as trusted key repository, in order to store the public keys of the members of its OSCORE groups, and provide such public keys to other members of the same group upon request. This specification recommends that the Group Manager is entrusted to perform this task.
- o Acting as network router device where endpoints register to correctly receive group messages sent to the multicast IP address of that group.
- o Autonomously and locally enforcing access policies to authorize new endpoints to join its OSCORE groups.

7. 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.

7.1. Group-level Security

The approach described in this document relies on commonly shared group keying material to protect communication within a group. This means that 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.

In addition, it is required that all group members are trusted, i.e. they do not forward the content of group messages 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).

7.2. 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 Case A is limited to requests, and same considerations hold.
- o Case B applies to all responses, and same considerations hold.

It follows that 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.

7.3. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide. However, this does not impair the security of the AEAD algorithm.

In fact, as long as the Master Secret is different for different groups and this condition holds over time, and as long as the Sender IDs within a group are unique, it follows that AEAD keys and nonces are different among different groups.

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, 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 the EIT-Digital High Impact Initiative ACTIVE.

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-13 (work in
progress), June 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>>.
- [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>>.
- [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-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-03 (work in progress), March 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-12 (work in progress), May 2018.
- [I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-01 (work in progress), March 2018.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-01 (work in progress), March 2018.
- [I-D.palombini-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-palombini-ace-key-groupcomm-01 (work in progress), June 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.
- [I-D.tiloca-ace-oscoap-joining]
Tiloca, M. and J. Park, "Joining OSCORE groups in ACE", draft-tiloca-ace-oscoap-joining-03 (work in progress), March 2018.
- [RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <<https://www.rfc-editor.org/info/rfc2093>>.

- [RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <<https://www.rfc-editor.org/info/rfc2094>>.
- [RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <<https://www.rfc-editor.org/info/rfc2627>>.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <<https://www.rfc-editor.org/info/rfc3376>>.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <<https://www.rfc-editor.org/info/rfc3740>>.
- [RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <<https://www.rfc-editor.org/info/rfc3810>>.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, DOI 10.17487/RFC4046, April 2005, <<https://www.rfc-editor.org/info/rfc4046>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, DOI 10.17487/RFC4535, June 2006, <<https://www.rfc-editor.org/info/rfc4535>>.
- [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>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [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>>.

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, e.g. by grouping lights in a building on a per floor basis.
- o Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, even when not registered as group member. In particular, communications with the Group Manager occurring during the join process to become a group member must also be secured.
- o Establishment and management of Security Contexts: an OSCORE Security Context must be established among the group members. In particular, a Common Context must be provided to a new joining endpoint together with a corresponding Sender Context. On the other hand, Recipient Contexts are locally and individually derived by each group member. 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 establishment and management of the Security Context is out of the scope of this document, and it is anticipated that an activity in IETF dedicated to the design of a generic key management scheme will include this feature, preferably based on [RFC3740][RFC4046][RFC4535].
- 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 endpoint. 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 sender endpoints.

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 impact on the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 upon completing each renewal of the Security Context and keying material in the group (see Section 2.1). In particular, once a new Master Secret has been distributed to the group, all the group members increment by 1 the Group Epoch in the Group Identifier of that group.

As an example, a 3-byte Group Identifier can be composed of: i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string; and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Security Common Context 61532 times in the group, its Group Identifier will assume value '0xb1f05c'.

As discussed in Section 7.3, 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 endpoint 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. 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 scope.

In order to receive multicast messages sent to the group, a joining endpoint has to register with a network router device [RFC3376][RFC3810], signaling its intent to receive packets sent to the multicast IP address of that group. As a particular case, the Group Manager can also act as such a network router device. Upon joining the group, endpoints are not required to know how many and what endpoints are active in the same group.

Furthermore, in order to participate in the secure group communication, an endpoint needs to be properly initialized upon joining the group. In particular, the Group Manager provides keying material and parameters to a joining endpoint, which can then initialize its own Security Context (see Section 2).

The following Appendix D.1 provides an example describing how such information can be provided to an endpoint upon joining a group through the responsible Group Manager. Then, Appendix D.2 discusses how public keys of group members can be handled and made available to group members. Finally, Appendix D.3 overviews how the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz] can be possibly used to support such a join process.

D.1. Join Process

An endpoint requests to join a group by sending a confirmable CoAP POST request to the Group Manager responsible for that group. This join request can reflect the format of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm]. Besides, it can be addressed to a CoAP resource associated to that group and carries the following information.

- o Group identifier: the Group Identifier (Gid) of the group, as known to the joining endpoint at this point in time. This may not fully coincide with the Gid currently associated to the group, e.g. if it includes a dynamic component. This information can be mapped to the first element of the 'scope' parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

- o Role: the exact role of the joining endpoint in the group. Possible values are: "client", "server", "silent server", "client and server", or "client and silent server". This information can be mapped to the second element of the 'scope' parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].
- o Retrieval flag: indication of interest to receive the public keys of the endpoints currently in the group, as included in the following join response. This flag must not be present if the Group Manager is not configured to store the public keys of group members, or if the joining endpoint is configured exclusively as silent server for the group to join. This information can be mapped to the 'get_pub_keys' parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].
- o Identity credentials: information elements to enforce source authentication of group messages from the joining endpoint, such as its public key. The exact content depends on whether the Group Manager is configured to store the public keys of group members. If this is the case, this information is omitted if it has been provided to the same Group Manager upon previously joining the same or a different group under its control. This information is also omitted if the joining endpoint is configured exclusively as silent server for the joined group. Appendix D.2 discusses additional details on provisioning of public keys and other information to enforce source authentication of joining endpoints's messages. This information can be mapped to the 'client_cred' parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].

The Group Manager must be able to verify that the joining endpoint is authorized to become a member of 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. Appendix D.3 describes how this can be achieved by leveraging the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

In case of successful authorization check, the Group Manager generates an Endpoint ID assigned to the joining endpoint, before proceeding with the rest of the join process. Instead, in case the authorization check fails, the Group Manager aborts the join process. Further details about the authorization of joining endpoint are out of scope.

As discussed in Section 2.1, it is recommended that the Security Context is renewed before the joining endpoint receives the group keying material and becomes a new active member of the group. This is achieved by securely distributing a new Master Secret and a new Group Identifier to the endpoints currently present in the same group.

Once renewed the Security Context in the group, the Group Manager replies to the joining endpoint with a CoAP response carrying the following information. This join response can reflect the format of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- o Security Common Context: the OSCORE Security Common Context associated to the joined group (see Section 2). This information can be mapped to the 'key' parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].
- o Endpoint ID: the Endpoint ID associated to the joining endpoint. This information is not included in case 'Role' in the join request is equal to "silent server". This information can be mapped to the 'clientID' parameter within the 'key' parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].
- o Member public keys: the public keys of the endpoints currently present in the group. This includes: the public keys of the non-silent servers currently in the group, if the joining endpoint is configured (also) as client; and the public keys of the clients currently in the group, if the joining endpoint is configured (also) as server or silent server. This information is omitted in case the Group Manager is not configured to store the public keys of group members or if the 'Retrieval flag' was not present in the join request. Appendix D.2 discusses additional details on provisioning public keys upon joining the group and on retrieving public keys of group members. This information can be mapped to the 'pub_keys' parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].
- o Group policies: a list of key words indicating the particular policies enforced in the group. This includes, for instance, the method to achieve synchronization of sequence numbers among group members (see Appendix E), as well as the rekeying protocol used to renew the keying material in the group (see Section 2.1). This information can be mapped to the 'group_policies' parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

- o Management keying material: the set of administrative keying material used to participate in the group rekeying process run by the Group Manager (see Section 2.1). The specific elements of this management keying material depend on the group rekeying protocol used in the group. For instance, this can simply consist in a group key encryption key and a pairwise symmetric key shared between the joining endpoint and the Group Manager, in case GKMP [RFC2093][RFC2094] is used. Instead, if key-tree based rekeying protocols like LKH [RFC2627] are used, it can consist in the set of symmetric keys associated to the key-tree leaf representing the group member up to the key-tree root representing the group key encryption key. This information can be mapped to the 'mgt_key_material' parameter of the Key Distribution Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm].

D.2. Provisioning and Retrieval of Public Keys

As mentioned in Section 6, it is recommended that the Group Manager acts as trusted key repository, so storing public keys of group members and providing them to other members of the same group upon request. In such a case, a joining endpoint provides its own public key to the Group Manager, as 'Identity credentials' of the join request, when joining the group (see Appendix D.1).

After that, the Group Manager should verify that the joining endpoint actually owns the associated private key, for instance by performing a proof-of-possession challenge-response, whose details are out of scope. In case of failure, the Group Manager performs up to a pre-defined maximum number of retries, after which it aborts the join process.

In case of successful challenge-response, the Group Manager stores the received public key as associated to the joining endpoint and its Endpoint ID. From then on, that public key will be available for secure and trusted delivery to other endpoints in the group. A possible approach for a group member to retrieve the public key of other group members is described in Section 7 of [I-D.palombini-ace-key-groupcomm].

Finally, the Group Manager sends the join response to the joining endpoint, as described in Appendix D.1.

The joining endpoint does not have to provide its own public key if that already occurred upon previously joining the same or a different group under the same Group Manager. However, separately for each group under its control, the Group Manager maintains an updated list

of active Endpoint IDs associated to the respective endpoint's public key.

Instead, in case the Group Manager does not act as trusted key repository, the following exchange with the Group Manager can occur during the join process.

1. The joining endpoint signs its own certificate by using its own private key. The certificate includes also the identifier of the issuer Certification Authority (CA). There is no restriction on the Certificate Subject included in the joining endpoint's certificate.
2. The joining endpoint specifies the signed certificate as 'Identity credentials' in the join request (Appendix D.1). The joining endpoint can optionally specify also a list of public key repositories storing its own certificate. In such a case, this information can be mapped to the 'pub_keys_repos' parameter of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].
3. When processing the join request, the Group Manager first validates the certificate by verifying the signature of the issuer CA, and then verifies the signature of the joining endpoint.
4. The Group Manager stores the association between the Certificate Subject of the joining endpoint's certificate and the pair {Group ID, Endpoint ID of the joining endpoint}. If received from the joining endpoint, the Group Manager also stores the list of public key repositories storing the certificate of the joining endpoint.

When a group member X wants to retrieve the public key of another group member Y in the same group, the endpoint X proceeds as follows.

1. The endpoint X contacts the Group Manager, specifying the pair {Group ID, Endpoint ID of the endpoint Y}.
2. The Group Manager provides the endpoint X with the Certificate Subject CS from the certificate of endpoint Y. If available, the Group Manager provides the endpoint X also with the list of public key repositories storing the certificate of the endpoint Y.
3. The endpoint X retrieves the certificate of the endpoint X from a key repository storing it, by using the Certificate Subject CS.

D.3. Group Joining Based on the ACE Framework

The join process to register an endpoint as a new member of a group can be based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], built on re-use of OAuth 2.0 [RFC6749].

In particular, the approach described in [I-D.tiloca-ace-oscoap-joining] uses the ACE framework to delegate the authentication and authorization of joining endpoints to an Authorization Server in a trust relation with the Group Manager. At the same time, it allows a joining endpoint to establish a secure channel with the Group Manager, by leveraging protocol-specific profiles of ACE, such as [I-D.ietf-ace-oscore-profile] and [I-D.ietf-ace-dtls-authorize], to achieve communication security, proof-of-possession and server authentication.

More specifically and with reference to the terminology defined in OAuth 2.0:

- o The joining endpoint acts as ACE Client;
- o The Group Manager acts as ACE Resource Server, with different CoAP resources for different groups it is responsible for;
- o An Authorization Server enables and enforces authorized access of the joining endpoint to the Group Manager and its CoAP resources paired with groups to join.

Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm]. Both the joining endpoint and the Group Manager have to adopt secure communication also for any message exchange with the Authorization Server. To this end, different alternatives are possible, such as OSCORE, DTLS [RFC6347] or IPsec [RFC4301].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by server endpoints to synchronize with 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 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 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 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 consistently with what specified in 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 4.2 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 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 group 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 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 sequence numbers lose synchronization. This can include a minimum gap between the sequence number of the latest accepted group request from a client and the 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 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 group 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 group 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 group 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 -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.2. 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.

Authors' Addresses

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

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

K. Lynn
P. van der Stok
Consultants
M. Koster
SmartThings
C. Amsuess
Energy Harvesting Solutions
July 02, 2018

CoRE Resource Directory: DNS-SD mapping
draft-ietf-core-rd-dns-sd-02

Abstract

Resource and service discovery are complimentary. Resource discovery provides fine-grained detail about the content of a 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 fields 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 January 3, 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	3
1.2. CoRE Resource Discovery	3
1.3. CoRE Resource Directories	4
1.4. DNS-Based Service Discovery	5
2. New Link-Format Attributes	6
2.1. Resource Instance attribute "ins"	6
2.2. Export attribute "exp"	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 Resource Type attribute "rt" to <ServiceType>	7
3.3. Domain mapping	8
3.4. TXT Record key=value strings	8
3.5. Importing resource links into DNS-SD	9
4. IANA considerations	9
5. Security considerations	9
6. References	10
6.1. Normative References	10
6.2. Informative References	11
Acknowledgments	11
Authors' Addresses	11

1. Introduction

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). 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].

Automated discovery of resources hosted by a constrained server is critical in M2M applications where human intervention is minimal and static interfaces result in brittleness. CoRE Resource Discovery is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other resource relations [RFC6690].

In contrast to resource discovery, service discovery generally refers to a coarser-grained resolution of an endpoint's IP address, port number, and protocol. This definition may be extended to include multi-function devices, where the result of the discovery process may include a path to a resource representing a RESTful service interface and possibly a reference to a description of the interface such as a JSON Hyper-Schema document [I-D.handrews-json-schema-hyperschema] per function.

Resource and service discovery are complimentary 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 (DNS-SD) [RFC6763] fields that permits discovery of CoAP services by either method. It also addresses the CoRE charter goal to interoperate with DNS-SD.

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 customary 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]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification also incorporates the terminology of [I-D.ietf-core-resource-directory].

1.2. CoRE Resource Discovery

[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) is typically a Universal Resource Identifier (URI [RFC3986]).

For example, "https://www.example.com/" has a "canonical" resource at "https://example.com", which has a "type" of "text/html".

The main function of Resource Discovery is to provide links for the resources hosted by the server, complemented by attributes about those resources and perhaps additional link relations. In CoRE this collection of links and attributes is itself a resource (as opposed to HTTP headers delivered with a specific resource).

[RFC6690] specifies a link format for use in CoRE Resource Discovery by extending the HTTP Link Header Format [RFC8288] to describe these link descriptions. The CoRE Link Format is carried as a payload and is assigned an Internet media type. CoRE Resource Discovery is accomplished by sending a GET request to the well-known URI `"/.well-known/core"`, which is defined as a default entry-point for requesting the collection of links about resources hosted by a server.

Resource Discovery can be performed either via 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 a URI for 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. A client would then match the appropriate Resource Type, Interface Description, and possible Content-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.

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 held on other servers (referred to as "end-points") and allows lookups to be performed for those resources. An endpoint is a web server associated with specific IP address and port; thus a physical device may host one or more endpoints. End-points may also act as clients.

The Resource Directory implements a set of REST interfaces for end-points to register and maintain collections of links, called resource directory entries. [I-D.ietf-core-resource-directory] specifies the web interfaces that an RD supports in order for web servers 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. Furthermore, new link attributes useful in conjunction with an RD are defined.

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; see [RFC6763] for a detailed specification.

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 host and the port of the 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 that identify the resources on an individual server or group of servers.

The <ServiceType> part is composed of at least 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 resources [ref?] might register the application protocol name "_oic", which all servers that advertise OCF resources would use as part of their ServiceType. The second label indicates the transport and is typically "_udp" for CoAP services. In cases where narrowing the scope of the search may be useful, these labels may be optionally preceded by a subtype name followed by the "_sub" label. An example of this more specific <ServiceType> is "light._sub._oic._udp".

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 SHOULD 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 could represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the service 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 [RFC6763]. The data consists 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). 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 useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

```
link-extension    = ( "ins" "=" (ptoken | quoted-string) )
                   ; The token or string is max 63 bytes
link-extension    = ( "exp" )
```

2.1. 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. This attribute is equivalent in use to the <Instance> portion of a DNS-SD record (see Section 1.4), and SHOULD be unique across resources with the same Resource Type attribute in the domain in which it is used. A Resource Instance SHOULD be a descriptive string like "Ceiling Light, Room 3", but MAY be a short ID like "AF39" or a unique UUID. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within the directory.

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].

2.2. 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].

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 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 Resource Type attribute "rt" to <ServiceType>

The <ServiceType> part of a DNS-SD Service Name is derived from the "rt" attribute and SHOULD conform to the reg-rel-type production of the Link Format defined in Section 2 of [RFC6690].

In practice, the ServiceType should unambiguously identify inter-operable devices. It is up to individual standards bodies to specify how to map between their registered Resource Type (rt=) values and ServiceType values. Two approaches are possible; either a

hierachical approach as in Section 1.4 above, or a flat identifier. Both approaches are shown below for illustration, but in practice only ONE would be specified.

In either case, the resulting application protocol name MUST be composed of at least a single Net-Unicode text string, without underscore '_' or 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 be mapped into "_oic-d-light._udp".

The application protocol name may be optionally followed by a period and a service subtype name consisting of a Net-Unicode text string, without underscore or period and limited to 63 bytes. This string is mapped to the DNS-SD <ServiceType> by appending a period followed by the "_sub" label and then appending a period followed by the service type label pair derived as in the previous paragraph. For example, rt="oic.d.light" might be mapped into "light._sub._oic._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 in which DNS zone the CoAP service should be registered. See, for example, Section 11 in [RFC6763].

3.4. TXT Record key=value strings

A number of [RFC6763] key/value pairs are derived from link-format information, to be exported in the DNS-SD as key=value strings in a 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. Importing 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 Resource Type as the <ServiceType>, and registered in the correct <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;rt="oic.d.light";ins="Spot";
  d="office";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a zone name "example.com" prefixed with "office":

```
_oic._udp.office.example.com      IN PTR
  Spot._oic._udp.office.example.com
light._sub._oic._udp.example.com  IN PTR
  Spot._oic._udp.office.example.com
Spot._oic._udp.office.example.com IN TXT
  txtver=1;path=/light/1
Spot._oic._udp.office.example.com IN SRV 0 0 5683
  node1.office.example.com.
node1.office.example.com.         IN AAAA FDFD::1234
```

In the above figure the Service Name is chosen as Spot._oic._udp.office.example.com without the light._sub service prefix. An alternative Service Name would be: Spot.light._sub._oic._udp.office.example.com.

4. IANA considerations

TBD

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, <<http://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, <<http://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, <<http://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, <<http://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>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<http://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, <<http://www.rfc-editor.org/info/rfc6335>>.

- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://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, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

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., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-14 (work in progress), July 2018.

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.

Authors' Addresses

Kerry Lynn
Consultant

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 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: January 3, 2019

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
July 02, 2018

CoRE Resource Directory
draft-ietf-core-resource-directory-14

Abstract

In many M2M 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 hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 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	4
2. Terminology	4
3. Architecture and Use Cases	7
3.1. Principles	7
3.2. Architecture	7
3.3. RD Content Model	9
3.4. Use Case: Cellular M2M	13
3.5. Use Case: Home and Building Automation	14
3.6. Use Case: Link Catalogues	14
4. Finding a Resource Directory	15
4.1. Resource Directory Address Option (RDAO)	17
5. Resource Directory	18
5.1. Payload Content Formats	19
5.2. URI Discovery	19
5.3. Registration	22
5.3.1. Simple Registration	27
5.3.2. Third-party registration	29
6. RD Groups	30
6.1. Register a Group	30
6.2. Group Removal	32
7. RD Lookup	33
7.1. Resource lookup	33
7.2. Lookup filtering	34
7.3. Resource lookup examples	36
8. Security Considerations	39
8.1. Endpoint Identification and Authentication	39
8.2. Access Control	40
8.3. Denial of Service Attacks	40
9. Authorization Server example	40
9.1. Registree-ep registers with RD	42
9.2. Third party Commissioning Tool (CT) registers registree-	

ep with RD.	42
9.3. Updating multiple links	43
10. IANA Considerations	43
10.1. Resource Types	43
10.2. IPv6 ND Resource Directory Address Option	44
10.3. RD Parameter Registry	44
10.3.1. Full description of the "Endpoint Type" Registration Parameter	46
10.4. "Endpoint Type" (et=) RD Parameter values	46
10.5. Multicast Address Registration	47
10.6. CBOR Web Token claims	47
11. Examples	48
11.1. Lighting Installation	48
11.1.1. Installation Characteristics	49
11.1.2. RD entries	50
11.2. OMA Lightweight M2M (LWM2M) Example	52
11.2.1. The LWM2M Object Model	53
11.2.2. LWM2M Register Endpoint	54
11.2.3. LWM2M Update Endpoint Registration	56
11.2.4. LWM2M De-Register Endpoint	56
12. Acknowledgments	56
13. Changelog	56
14. References	62
14.1. Normative References	63
14.2. Informative References	63
Appendix A. Registration Management	65
A.1. Registration Update	66
A.2. Registration Removal	69
A.3. Read Endpoint Links	70
A.4. Update Endpoint Links	71
A.5. Endpoint and group lookup	71
Appendix B. Web links and the Resource Directory	73
B.1. A simple example	73
B.1.1. Resolving the URIs	73
B.1.2. Interpreting attributes and relations	74
B.2. A slightly more complex example	74
B.3. Enter the Resource Directory	75
B.4. A note on differences between link-format and Link headers	76
Appendix C. Syntax examples for Protocol Negotiation	77
Appendix D. Modernized Link Format parsing	78
D.1. For endpoint developers	79
Authors' Addresses	79

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g., 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at 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 [RFC5988]. 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 M2M 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 hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resource descriptions. Furthermore, new link 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], [RFC5988] 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 resolving an absolute URI against any base URI gives the original URI, and that resolving an empty URI reference gives the base URI.

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.

Group

A group in the Resource Directory specifies a set of endpoints that are enabled with the same multicast address for the purpose of efficient group communications. All groups within a sector have unique names.

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 by the Endpoint at registration time, and is used by the Resource Directory to resolve relative references inside the registration into absolute 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.

Group Resource

A resource in the RD containing registration resources of the Endpoints that form a group.

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.

Registree-ep

Registree-ep is the endpoint that is registered into the RD. The registree-ep can register itself, or a CT registers the registree-ep.

RDAO

Resource Directory Address Option.

For several operations, interface descriptions are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Those templates contain normative content in their Interaction, Method, URI Template and URI Template Variables sections 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 the implementing parties should be prepared to 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.

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 device, or across boundaries that would be limiting those operations.

It provides a cache (in the high-level sense, not as defined in [RFC7252]/[RFC2616]) of data that could otherwise only be obtained by directly querying the `/.well-known/core` resource on the target device, or by accessing those resources with a multicast request.

From that, it follows that only information should be stored in the resource directory that is discovered from querying the described device's `/.well-known/core` resource directly.

It also follows that data in the resource directory can only be provided by the device whose descriptions are cached or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf of agents too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory. Changes in the Resource Directory do not propagate automatically back to the web server from where the links originated.

3.2. Architecture

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are 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 sets of Web Links (called resource directory registration entries), and for clients to lookup resources from the RD or maintain groups. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Sectors. The set of endpoints grouped for group communication can be defined by the RD or configured by a Commissioning Tool. This information hierarchy is shown in Figure 2.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Endpoints proactively register and maintain resource directory registration entries on the RD, which are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a resource directory registration entry. It is also possible for an RD to fetch Web Links from endpoints and add them as resource directory registration entries.

At the first registration of a set of entries, 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 registration entries.

A lookup interface for discovering any of the Web Links held in the RD is provided using the CoRE Link Format.

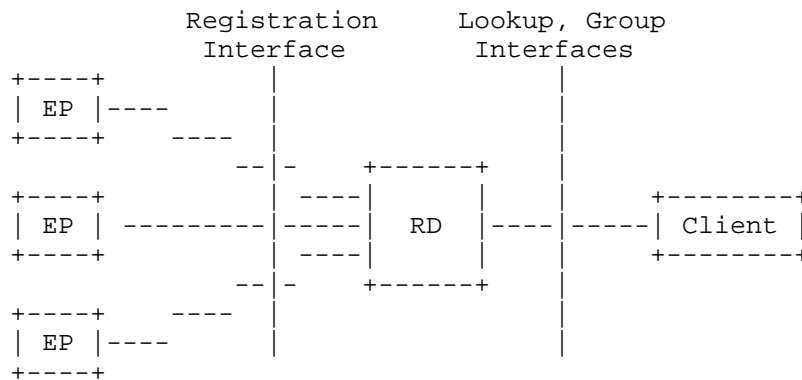


Figure 1: The resource directory architecture.

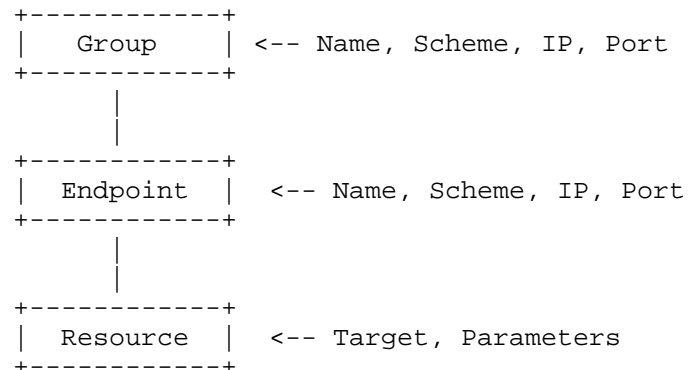


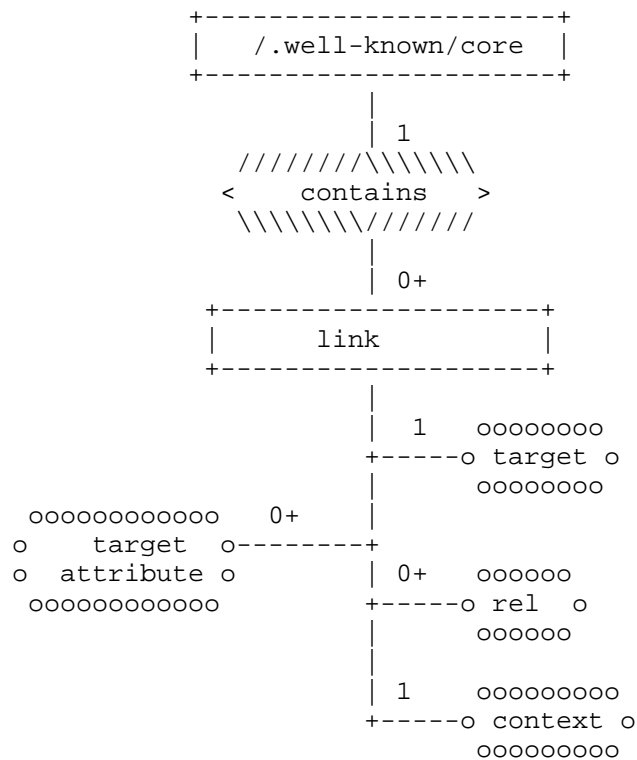
Figure 2: The resource directory information hierarchy.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 3 and Figure 4 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 3: E-R Model of the content of `/.well-known/core`

The model shown in Figure 3 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 [RFC5988]):

- 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, eg. `_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 (eg. `_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 (eg. resource type (rt), interface (if), or content-type (ct)). These provide additional information about the target URI.

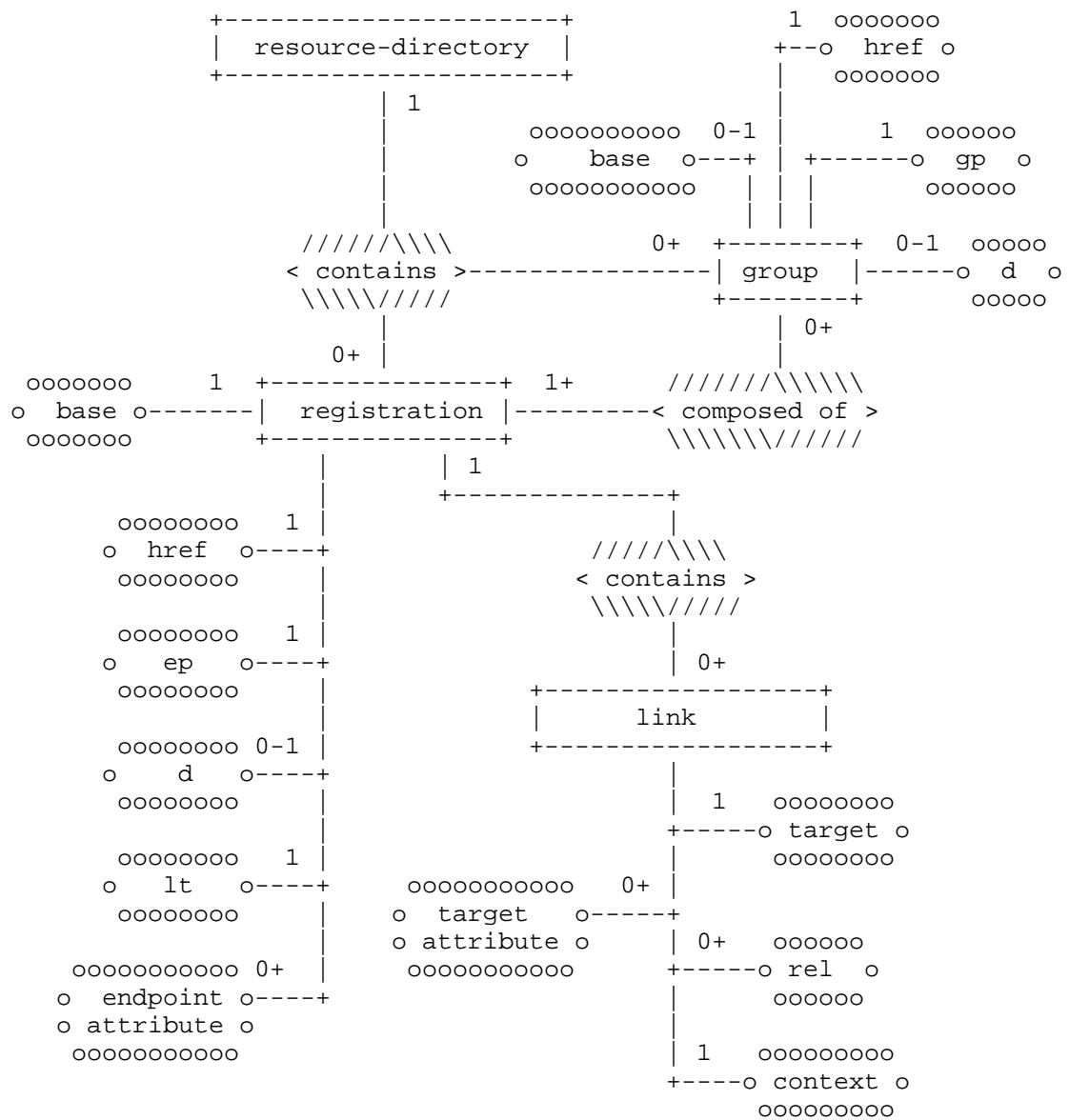


Figure 4: E-R Model of the content of the Resource Directory

The model shown in Figure 4 models the contents of the resource directory which contains in addition to /.well-known/core:

- o 0 to n Registration (entries) of endpoints,

- o 0 or more Groups

A Group has:

- o a group name ("gp"),
- o optionally a sector (abbreviated "d" for historical reasons),
- o a group resource location inside the RD ("href"),
- o zero or one multicast addresses expressed as a base URI ("base"),
- o and is composed of zero or more registrations (endpoints).

A registration is associated with one endpoint. A registration can be part of 0 or more Groups. A registration defines a set of links as defined for /.well-known/core. A Registration has six types of attributes:

- o a unique endpoint name ("ep")
- 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")
- o optional additional endpoint attributes (from Section 10.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (eg. [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 3.

3.4. 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.

3.5. 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.

3.6. 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 the 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] are supplied by Resource Directories, which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links. 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. Groups may be defined to support efficient data transport.

4. Finding a Resource Directory

A (re-)starting device may want to find one or more resource directories to make itself known with.

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 a resource-record type to look up under this name; 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.
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.

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.

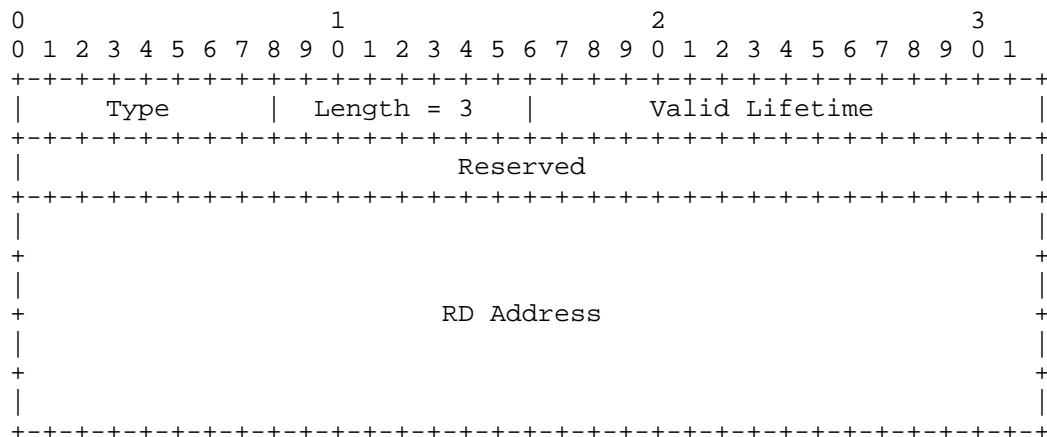
If multiple candidate addresses are discovered, the device may pick any of them initially, unless the discovery method indicates a more precise selection scheme.

4.1. 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 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 5: Resource Directory Address Option

5. Resource Directory

This section defines the required set of REST interfaces between a Resource Directory (RD) and endpoints. Although the examples throughout this section assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. In all definitions in this section, 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 defined in this section.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

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.

5.1. 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 MUST have an equivalent serialization in the application/link-format content format.

5.2. 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.

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, and `"core.rd-group"` is used to discover the URI path for RD Group 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.

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the `"ct"` link 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. Links to Resource Directories MAY be registered in other Resource Directories. The well-known entry points SHOULD be provided to enable the bootstrapping of 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 SHOULD therefore accept URIs of all schemes they support, both in absolute and relative forms, 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 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 (optional). MAY contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", "core.rd-lookup-gp", "core.rd-group" or "core.rd*"

Content-Format: application/link-format (if any)

Content-Format: application/link-format+json (if any)

Content-Format: application/link-format+cbor (if any)

The following response codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format, application/link-format+json, or application/link-format+cbor

payload containing one or more matching entries for the RD resource.

Failure: 4.00 "Bad Request" or 400 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

HTTP support : YES (Unicast only)

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource is, in this example, at /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 resource paths.

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,
</rd-lookup/gp>;rt="core.rd-lookup-gp";ct=40,
</rd-group>;rt="core.rd-group";ct=40

Figure 6: 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 the CBOR and JSON representation of link format. The RD resource paths /rd, /rd-lookup, and /rd-group 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 these and later occurrences of TBD64 and TBD504 with the numeric ID values assigned by IANA to application/link-format+cbor and application/link-format+json, respectively, as they are defined in I-D.ietf-core-links-json.]

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",  
</rd-lookup/gp>;rt="core.rd-lookup-gp";ct="40 TBD64 TBD504",  
</rd-group>;rt="core.rd-group";ct="40 TBD64 TBD504"
```

From a management and maintenance perspective, it is necessary to identify the components that constitute the 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 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.

5.3. Registration

After discovering the location of an RD, a registree-ep or CT MAY register the resources of the registree-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], JSON CoRE Link Format (application/link-format+json), or CBOR CoRE Link Format (application/link-format+cbor) [I-D.ietf-core-links-json], 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 10.3). 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 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 an update identified by a given (ep, d) value pair:

- o when the parameter values of the Update generate the same attribute values as already present, the location of the already existing registration is returned.
- o when for a given (ep, d) value pair the update generates attribute values which are different from the existing one, the existing registration is removed and a new registration with a new location is created.
- o when the (ep, d) value pair of the update is different from any existing registration, a new registration is generated.

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 network address.

Link format documents submitted to the resource directory are interpreted as Modernized Link Format (see Appendix D) by the RD. A registree-ep SHOULD NOT submit documents whose interpretations according to [RFC6690] and Appendix D differ and RFC6690 interpretation is intended to avoid the ambiguities described in Appendix B.4.

In practice, most links (precisely listed in Appendix D.1) can be submitted without consideration for those details.

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. The maximum length of this parameter is 63 bytes. If the RD is configured to recognize the endpoint (eg. based on its security context), the endpoint sets no endpoint name, and the RD assigns one based on a set of configuration parameter values.

d := Sector (optional). The sector to which this endpoint belongs. The maximum length of this parameter is 63 bytes. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector or leave it empty. 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 request's links 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. This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the endpoint uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

If the endpoint which is located behind a NAT gateway is registering with a Resource Directory which is on the network service side of the NAT gateway, the endpoint MUST use a persistent port for the outgoing registration in order to provide the NAT gateway with a valid network address for replies and incoming requests.

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; they can submit payloads for interpretation as Modernized Link Format. Typically, links submitted by such an endpoint are of the "path-noscheme" (starts with a path not preceded by a slash, precisely defined in [RFC3986] Section 3.3) form.

extra-attrs := Additional registration attributes (optional).
The endpoint can pass any parameter registered at Section 10.3 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

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for 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.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

If the registration fails with a Service Unavailable response and a Max-Age option or Retry-After header, the registering endpoint SHOULD retry the operation after the time indicated. If the registration fails in another way, including request timeouts, or if the Service Unavailable error persists after several retries, or indicates a longer time than the endpoint is willing to wait, 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, eg. 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 registree-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 6.

```
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 7: 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 and the JSON Link Format.

```
Req: POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: example.com
Content-Type: application/link-format+json
Payload:
[
  {"href": "/sensors/temp", "ct": "41", "rt": "temperature-c",
   "if": "sensor", "anchor": "coap://spurious.example.com:5683"},
  {"href": "/sensors/light", "ct": "41", "rt": "light-lux",
   "if": "sensor"}
]

Res: 201 Created
Location: /rd/4521
```

5.3.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5.3. 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 registree-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 D).

The registree-ep then finds one or more addresses of the directory server as described in Section 4.

The registree-ep finally asks the selected directory server to probe it for resources and publish them as follows:

The registree-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 registree-ep's default discovery URI to obtain the link-format payload to register.

The registree-ep includes the same registration parameters in the POST request as it would per Section 5.3. The registration base URI of the registration is taken from the requesting server's URI.

The Resource Directory **MUST NOT** query the registree-ep's data before sending the response; this is to accommodate very limited endpoints.

The success condition only indicates that the request was valid (ie. the passed parameters are valid per se), not that the link data could be obtained or parsed or was successfully registered into the RD.

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 5.3. 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 codes are defined for this interface:

Success: 2.04 "Changed".

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

HTTP support: NO

For the second interaction triggered by the above, the registree-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 codes are defined for this interface:

Success: 2.05 "Content".

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.04 "Not Found". /.well-known/core does not exist or is empty.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

HTTP support: NO

The registration resources MUST be deleted after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

The following example shows a registree-ep using Simple Registration, by simply sending an empty POST to a resource directory.

```
Req:(to RD server from [2001:db8:2::1])
POST /.well-known/core?lt=6000&ep=node1
No payload
```

Res: 2.04 Changed

(later)

```
Req: (from RD server to [2001:db8:2::1])
GET /.well-known/core
Accept: 40
```

```
Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

5.3.2. 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. The commissioning tool can fill the Resource Directory from a database or other means. For that purpose the scheme, IP address and port of the registered device is indicated in the "base" parameter of the registration described in Section 5.3.

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.

6. RD Groups

This section defines the REST API for the creation, management, and lookup of endpoints for group operations. Similar to endpoint registration entries in the RD, groups may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

6.1. Register a Group

In order to create a group, a commissioning tool (CT) used to configure groups, makes a request to the RD indicating the name of the group to create (or update), optionally the sector the group belongs to, and optionally the multicast address of the group. This specification does not require that the endpoints belong to the same sector as the group, but a Resource Directory implementation can impose requirements on the sectors of groups and endpoints depending on its configuration.

The registration message is a list of links to registration resources of the endpoints that belong to that group. The CT can use any URI reference discovered using endpoint lookup from the same server or obtained by registering an endpoint using third party registration and enter it into a group. The use of other URIs is not specified in this document and can be defined in others.

The commissioning tool SHOULD not send any target attributes with the links to the registration resources, and the resource directory SHOULD reject registrations that contain links with unprocessable attributes.

Configuration of the endpoints themselves is out of scope of this specification. Such an interface for managing the group membership of an endpoint has been defined in [RFC7390].

The registration request interface is specified as follows:

Interaction: CT -> RD

Method: POST

URI Template: {+rd-group}{?gp,d,base}

URI Template Variables:

`rd-group` := RD Group URI (mandatory). This is the location of the RD Group REST API.

`gp` := Group Name (mandatory). The name of the group to be created or replaced, unique within that sector. The maximum length of this parameter is 63 bytes.

`d` := Sector (optional). The sector to which this group belongs. The maximum length of this parameter is 63 bytes. When this parameter is not present, the RD MAY associate the group with a configured default sector or leave it empty.

`base` := Group Base URI (optional). This parameter sets the scheme, address and port of the multicast address associated with the group. When `base` is used, scheme and host are mandatory and port parameter is optional.

Content-Format: application/link-format

Content-Format: application/link-format+json

Content-Format: application/link-format+cbor

The following response codes are defined for this interface:

Success: 2.01 "Created" or 201 "Created". The Location header or Location-Path option MUST be returned in response to a successful group CREATE operation. This location MUST be a stable identifier generated by the RD as it is used for delete operations of the group resource.

As with the Registration operation, the location MUST NOT have a query or fragment component.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following example shows an EP registering a group with the name "lights" which has two endpoints. The RD group path `/rd-group` is an example RD location discovered in a request similar to Figure 6.

```
Req: POST coap://rd.example.com/rd-group?gp=lights
      &base=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
</rd/4521>,
</rd/4522>

Res: 2.01 Created
Location-Path: /rd-group/12
```

A relative href value denotes the path to the registration resource of the Endpoint. When pointing to a registration resource on a different RD, the href value is an absolute URI.

6.2. Group Removal

A group can be removed simply by sending a removal message to the location of the group registration resource which was returned when initially registering the group. Removing a group **MUST NOT** remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the path of the group resource returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Group does not exist.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples shows successful removal of the group from the RD with the example location value /rd-group/12.

Req: DELETE /rd-group/12

Res: 2.02 Deleted

7. 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. Atom or HTML Link) or using more advanced interfaces (e.g. supporting context or semantic based lookup).

RD Lookup allows lookups for groups, 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, a group lookup MUST return a list of groups, 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
Group	core.rd-lookup-gp	Optional

Table 1: Lookup Types

7.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD if they were accessed on the endpoint itself. The links and link parameters returned are equal to the submitted, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the (explicitly or implicitly set) base URI of the registration

as the anchor. Links whose href or anchor was submitted as an absolute URI are returned with respective attributes unmodified.

Above rules allow the client to interpret the response as links without any further knowledge of what the RD does. 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.

7.2. 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 its alternate content-formats ("application/link-format+json" or "application/link-format+cbor").

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]; eg. "?if=core.s" matches ";if=abc core.s;"). A link also matches a search criterion if the link that would be produced for any of its containing entities would match the criterion, or an entity contained in it would: A search criterion matches an endpoint if it matches the endpoint itself, any of the groups it is contained in or any resource it contains. A search criterion matches a resource if it matches the resource itself, the resource's endpoint, or any of the endpoint's groups.

Note that "href" is also 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, or any group resource that endpoint is contained in. Queries for resource link targets MUST be in absolute form and are matched against a

resolved link target. Queries for groups and endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in absolute form otherwise; the RD SHOULD recognize either.

Clients 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 5.2.

search := Search criteria for limiting the number of results (optional).

page := Page (optional). Parameter can not 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.

Content-Format: application/link-format (optional)

Content-Format: application/link-format+json (optional)

Content-Format: application/link-format+cbor (optional)

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload containing matching entries for the lookup. The payload can contain zero links (which is an empty payload, "80" (hex) or "[]" in the respective content format), indicating that no entities matched the request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The group and endpoint lookup return registration resources which can only be manipulated by the registering endpoint. Examples of group and endpoint lookup belong to the management aspects of the RD and are shown in Appendix A.5. The resource lookup examples are shown in this section.

7.3. 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 6:

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"

The same lookup using the CBOR Link Format media type:

Req: GET /rd-lookup/res?rt=temperature
Accept: TBD64

Res: 2.05 Content
Content-Format: TBD64
Payload in Hex notation:
81A3017823636F61703A2F2F5B323030313A6462383A333A3A3132335D3A363136313
62F74656D7003781E636F61703A2F2F5B323030313A6462383A333A3A3132335D3A36
31363136096B74656D7065726174757265
Decoded payload:
[{1: "coap://[2001:db8:3::123]:61616/temp", 9: "temperature",
3: "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"
```

8. Security Considerations

The security considerations as described in Section 7 of [RFC5988] 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 is determined to be unique within (the sector of) an RD by the Endpoint identifier parameter included during Registration, and any associated TLS or DTLS security bindings. 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 or Client 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 managed by 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 puts the endpoint name of device B. 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 9 specifies an example that removes this threat by using an Authorization Server for endpoints that have a certificate installed.

8.2. Access Control

Access control SHOULD be performed separately for the RD registration, Lookup, and group 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. Authorization Server example

When threats may occur as described in Section 8.1, an Authorization Server (AS) as specified in [I-D.ietf-ace-oauth-authz] can be used to remove the threat. An authorized registry request to the Resource

Directory (RD) is accompanied by an Access Token that validates the access of the client to the RD. In this example, the contents of the Access Token is specified by a CBOR Web Token (CWT) [RFC8392]. Selecting one of the scenarios of [I-D.ietf-anima-bootstrapping-keyinfra], the registree-ep has a certificate that has been inserted at manufacturing time. The contents of the certificate will be used to generate the unique endpoint name. The certificate is uniquely identified by the leftmost CNcomponent of the subject name appended with the serial number. The unique certificate identifier is used as the unique endpoint name. The same unique identification is used for the registree-ep and the Commissioning Tool.

The case of using RPK or PSK is outside the scope of this example.

Figure 8 shows the example certificate used to specify the claim values in the CWT. Serial number 01:02:03:04:05:06:07:08, and CN field, Fairhair, in the subject field are concatenated to create a unique certificate identifier: Fairhair-01:02:03:04:05:06:07:08, which is used in Figure 9 and Figure 10 as "sub" claim and "epn" claim values respectively.

Certificate: Data:

```

  Version: 3 (0x2)
  Serial Number: 01:02:03:04:05:06:07:08
  Signature Algorithm: md5WithRSA
  Encryption Issuer: C=US, ST=Florida, O=Acme, Inc., OU=Security,
                                     CN=CA
  Authority/emailAddress=ca@acme.com
  Validity Not Before: Aug 20 12:59:55 2013 GMT
                                     Not After : Aug 20 12:59:55 2013 GMT
  Subject: C=US, ST=Florida, O=Acme, Inc., OU=Sales, CN=Fairhair
  Subject Public Key
  Info: Public Key Algorithm: rsaEncryption
  RSA Public Key: (1024 bit) Modulus (1024 bit):
00:be:5e:6e:f8:2c:c7:8c:07:7e:f0:ab:a5:12:db:
fc:5a:1e:27:ba:49:b0:2c:e1:cb:4b:05:f2:23:09:
77:13:75:57:08:29:45:29:d0:db:8c:06:4b:c3:10:
88:e1:ba:5e:6f:1e:c0:2e:42:82:2b:e4:fa:ba:bc:
45:e9:98:f8:e9:00:84:60:53:a6:11:2e:18:39:6e:
ad:76:3e:75:8d:1e:b1:b2:1e:07:97:7f:49:31:35:
25:55:0a:28:11:20:a6:7d:85:76:f7:9f:c4:66:90:
e6:2d:ce:73:45:66:be:56:aa:ee:93:ae:10:f9:ba:
24:fe:38:d0:f0:23:d7:a1:3b
  Exponent: 65537 (0x10001)

```

Figure 8: Sample X.509 version 3 certificate for Fairhair device issued by the Acme corporation.

Three sections for as many authorized RD registration scenarios describe: (1) the registree-ep registers itself with the RD, (2) a 3rd party Commissioning Tool (CT) registers the registree-ep with the RD, and (3) A client updates multiple links in an RD.

9.1. Registree-ep registers with RD

The registree-ep sends a Request to the RD accompanied by a CBOR Web Token (CWT). To prevent ambiguities, the URI of the authorized request cannot contain the ep= or the d= parameters which are specified in the CWT. When these parameters are present in the URI, the request is rejected with CoAP response code 4.00 (bad request). The CWT of Figure 9 authorizes the registree-ep to register itself in the RD by specifying the certificate identifier of the registree-ep in the sub claim. The same value is assigned to the endpoint name of the registree-ep in the RD.

The claim set of the CWT is represented in CBOR diagnostic notation

```
{
  /iss/ 1: "coaps://as.example.com",    /identifies the AS/
  /sub/ 2: "Fairhair_01:02:03:04:05:06:07:08",
      / certificate identifier uniquely identifies registree-ep/
  /aud/ 3: "coaps://rd.example.com"    / audience is the RD/
}
```

Figure 9: Claim set of CWT for registering registree-ep

9.2. Third party Commissioning Tool (CT) registers registree-ep with RD.

The CT sends a Request to the RD accompanied by a CBOR Web Token (CWT). To prevent ambiguities, the URI of an authorized request cannot contain the ep= or the d= parameters which are specified in the CWT. When these parameters are present in the URI, the request is rejected with CoAP response code 4.00 (bad request). The CWT of Figure 10 authorizes the CT to register the registree-ep by specifying the certificate identifier, Fairhair_08:07:06:05:04:03:02:01, of the CT in the "sub" claim. Next to the certificate identifier of the CT, the CWT needs to specify the security identifier of the registree-ep. The new "rd_epn" claim is used to specify the value of the certificate identifier Fairhair_01:02:03:04:05:06:07:08, of the registree-ep. The CWT may contain the optional new "rd_sct" claim to assign a sector name to the registree-ep.

The claim set is represented in CBOR diagnostic notation

```
{
  /iss/      1: "coaps://as.example.com",    / identifies the AS/
  /sub/      2: "Fairhair_08:07:06:05:04:03:02:01",
              / certificate identifier uniquely identifies CT/
  /aud/      3: "coaps://rd.example.com",    / audience is the RD/
  /rd_epn/   y: "Fairhair_01:02:03:04:05:06:07:08",
              /certificate identifier uniquely identifies registree-ep/
  /rd_sct/   z: "my-devices"                /optional sector name/
}
```

Figure 10: Claim set of CWT for registering registree-ep by CT

9.3. Updating multiple links

Appendix A.4 of RD specifies that multiple links can be updated with a media format to be specified. The updating endpoint sends a Request to the RD accompanied by a CWT. The "sub" claim of the CWT contains the certificate identifier of the updating endpoint. Updating registrations and links cannot not change or delete the endpoint names. Consequently, the updating endpoint is authorized by the CWT to change all links of its registrations but cannot delete or add registrations. The CWT of Figure 9 and Figure 10 authorize an updating registree-ep or an updating CT respectively.

10. IANA Considerations

10.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values subregistry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 5.2
core.rd-group	Group directory resource of an RD	RFCTHIS Section 5.2
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 5.2
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 5.2
core.rd-lookup-gp	Group lookup of an RD	RFCTHIS Section 5.2
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 7
core.rd-gp	Group resource of an RD	RFCTHIS Section 7

10.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the subregistry "IPv6 Neighbor Discovery Option Formats":

- o Resource Directory address Option (38)

10.3. 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 link attributes,
- o indication of whether it can be passed as a query parameter at registration of endpoints or groups, as a query parameter in lookups, or be expressed as a link attribute,
- o validity requirements if any, and
- o a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a parmname as used in [RFC5988].

The description must give details on which registrations they apply to (Endpoint, group registrations or both? Can they be updated?), and how they are 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		RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Group Name	gp	Integer	RLA	Name of a group in the RD
Page Count	page count		L	Used for pagination
Endpoint Type	et		L RLA	Used for pagination Semantic name of the endpoint (see Section 10.4)

Table 2: RD Parameters

(Short: Short name used in query parameters or link attributes. Use: R = used at registration, L = used at lookup, A = expressed in link 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 (Eg. 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 link 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] link attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

10.3.1. 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 subregistry. 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.

10.4. "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 10.3.1.
- 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 is initially empty.

10.5. 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 FFOX::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.

10.6. CBOR Web Token claims

This specification registers the following new claims in the CBOR Web Token (CWT) registry of CBOR Web Token Claims:

Claim "rd_epn"

- o Claim Name: "rd_epn"
- o Claim Description: The endpoint name of the RD entry as described in Section 9 of RFC7255.
- o JWT Claim Name: N/A

- o Claim Key: y
- o Claim Value Type(s): 0 (uint), 2 (byte string), 3 (text string)
- o Change Controller: IESG
- o Specification Document(s): Section 9 of RFC7238

Claim "rd_sct"

- o Claim Name: "rd_sct"
- o Claim Description: The sector name of the RD entry as described in Section 9 of RFC7238.
- o JWT Claim Name: N/A
- o Claim Key: z
- o Claim Value Type(s): 0 (uint), 2 (byte string), 3 (text string)
- o Change Controller: IESG
- o Specification Document(s): Section 9 of RFC7238

Mapping of claim name to CWT key

Parameter name	CBOR key	Value type
rd_epn	y	Text string
rd_sct	z	Text string

11. Examples

Two examples are presented: a Lighting Installation example in Section 11.1 and a LWM2M example in Section 11.2.

11.1. 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. 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.

11.1.1.1. 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 11.1.2 the use of resource directory during installation is presented.

11.1.2. 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 5.2 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, these two endpoints and the endpoint of the presence sensor are registered as members of the group.

```
Req: POST coap://[2001:db8:4::ff]/rd-group
?gp=grp_R2-4-015&base=coap://[ff05::1]
Payload:
</rd/4521>,
</rd/4522>,
</rd/4523>
```

```
Res: 2.01 Created
Location-Path: /rd-group/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 own IPv6 address, looks up the groups containing light resources it is assigned to:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/gp
?d=R2-4-015&base=coap://[2001:db8:4::1]&rt=light
```

```
Res: 2.05 Content
</rd-group/501>;gp="grp_R2-4-015";base="coap://[ff05::1]"
```

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.

11.2. 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-group or rd-lookup interfaces.

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.

11.2.1. 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).

11.2.2. 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.

11.2.3. LWM2M Update Endpoint Registration

The Lwm2m update is really very similar to the registration update as described in Appendix A.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.

11.2.4. 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 Appendix A.2.

12. 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, and Jan Newmarch 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.

13. Changelog

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.

14. References

14.1. Normative References

- [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.
- [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>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<https://www.rfc-editor.org/info/rfc5988>>.
- [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>>.

14.2. 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.arkko-core-dev-urn]
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-05 (work in progress), October 2017.
- [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.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-12 (work in progress), May 2018.
- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-16 (work in progress), June 2018.
- [I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-08 (work in progress), March 2018.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

Appendix A. Registration Management

This section describes how the registering endpoint can maintain the registries that it created. The registering endpoint can be the registree-ep or the CT. An endpoint SHOULD NOT use this interface for registries that it did not create. The registries 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 and remove it from any group it belongs to. If the Registration Resource is removed, the corresponding endpoint will need to be re-registered.

The Registration Resource may also be used to inspect the registration resource using GET, update the registration, cancel the registration using DELETE, do an endpoint lookup, or a group lookup.

These operations are described below.

A.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 context- registration parameters "lt", "base" as in Section 5.3. 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 5.3.

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 context of the registration is changed in an update explicitly or implicitly, relative references submitted in the original registration or later updates are resolved anew against the new context (like in the original registration).

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 Appendix A.4).

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,con,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 context 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 links of the registration, following the same restrictions as in the registration. If the parameter is not set and was set explicitly before, the previous Base URI value is kept unmodified. If the parameter is not set and was not set explicitly 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 response codes are defined for this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

If the registration update fails with a "Service Unavailable" response and a Max-Age option or Retry-After header, the registering endpoint SHOULD retry the operation after the time indicated. If the registration fails in another way, including request timeouts, or if the time indicated exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows 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 context (con)=coap://local-proxy-old.example.com:5683
- o payload of Figure 7

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 context to "coaps://new.example.com:5684":

Req: POST /rd/4521?con=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",
```

A.2. Registration Removal

Although RD entries 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.

Removed registrations are implicitly removed from the groups to which they belong.

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 response codes are defined for this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

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

A.3. Read Endpoint Links

Some registering endpoints may wish to manage their links as a collection, and may need to read the current set of links stored in the registration resource, in order to determine link maintenance operations.

One or more links MAY be selected by using query filtering as specified in [RFC6690] Section 4.1

If no links are selected, the Resource Directory SHOULD return an empty payload.

The read request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: {+location}{?href,rel,rt,if,ct}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

href,rel,rt,if,ct := link relations and attributes specified in the query in order to select particular links based on their relations and attributes. "href" denotes the URI target of the link. See [RFC6690] Sec. 4.1

The following response codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" upon success with an "application/link-format", "application/link-format+cbor", or "application/link-format+json" payload.

Failure: 4.00 "Bad Request" or 400 "Bad Request". Malformed request.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have expired).

Failure: 5.03 "Service Unavailable" or 503 "Service Unavailable". Service could not perform the operation.

HTTP support: YES

The following examples show successful read of the endpoint links from the RD, with example location value /rd/4521 and example registration payload of Figure 7.

Req: GET /rd/4521

Res: 2.01 Content

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"
```

A.4. Update Endpoint Links

An iPATCH (or PATCH) update ([RFC8132]) can add, remove or change the links of a registration.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

A.5. Endpoint and group lookup

Endpoint and group lookups result in links to registration resources and group resources, respectively. Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as link attributes to their endpoint link unless their specification says otherwise.

Group resources are annotated with their group names (gp), sector (d, if present) and multicast address (base, if present) as well as a constant resource type (rt="core.rd-gp").

Serializations derived from Link Format, SHOULD present links to groups and endpoints in path-absolute form or, if required, as absolute references. (This approach avoids the RFC6690 ambiguities.)

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 or groups in lookup that are not hosted at the same address. Lookup clients MUST be prepared to see arbitrary URIs as registration or group resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

For groups, a Resource Directory as specified here does not provide a lookup mechanism for the resources that can be accessed on a group's multicast address (ie. no lookup will return links like "<coap://[ff35:30:2001:db8::1]/light>;..." for a group registered with "base=coap://[ff35...]"). Such an additional lookup interface could be specified in an extension document.

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",
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";
et="oic.d.sensor";ct="40";d="floor-3"
```

The following example shows a client performing a group lookup for all groups:

```
Req: GET /rd-lookup/gp
```

```
Res: 2.05 Content
</rd-group/1>;gp="lights1";d="example.com";
    base="coap://[ff35:30:2001:db8::1]",
</rd-group/2>;gp="lights2";d="example.com";
    base="coap://[ff35:30:2001:db8::2]"
```

The following example shows a client performing a lookup for all groups the endpoint "node1" belongs to:

```
Req: GET /rd-lookup/gp?ep=node1
```

```
Res: 2.05 Content
</rd-group/1>;gp="lights1"
```

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.

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 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. As long as all involved links follow the restrictions set forth for this document (see Appendix B.4), the base URI to resolve this against the requested URI.

The URI of the requested resource can be composed 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".

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 type is `text/plain (ct=0)`.

A relation in a web link is a three-part statement that the context resource has a named relation to the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In [RFC6690] link-format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context of the link is the URI of the requested document itself. 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:

```
</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 document's Base URI 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 document Base URI any more, but about the target and that address.

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"`.

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]/`" because that is the address the registration was sent from (and no explicit "`con=`" was given).

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 would contain the (almost) same 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 is subject of ongoing discussion about RFC6690).

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 registered with an explicit context (eg. "?ep=simple-host1&con=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 [RFC5988], 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"). [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.

In the context of a Resource Directory, the authors decided not to let this become an issue by requiring that RFC6690 links be serialized in a way that either rule set can be applied and give

the same results. Note that all examples of [RFC6690], [RFC8288] and this document comply with that rule.

The Modernized Link Format is introduced in Appendix D to formalize what it means to apply the ruleset of RFC8288 to Link Format documents.

- 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. Syntax examples for Protocol Negotiation

[This appendix should not show up in a published version of this document.]

The protocol negotiation that is being worked on in [I-D.silverajan-core-coap-protocol-negotiation] makes use of the Resource Directory.

Until that document is update to use the latest resource-directory specification, here are some examples of protocol negotiation with the current Resource Directory:

An endpoint could register as follows from its address
"[2001:db8:f1::2]:5683":

```
Req: POST coap://rd.example.com/rd?ep=node1
      &at=coap+tcp://[2001:db8:f1::2]
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"
```

```
Res: 2.01 Created
Location-Path: /rd/1234
```

An endpoint lookup would just reflect the registered attributes:

```
Req: GET /rd-lookup/ep
```

```
Res: 2.05 Content
</rd/1234>;ep="node1";con="coap://[2001:db8:f1::2]:5683";
  at="coap+tcp://[2001:db8:f1::2]"
```

A UDP client would then see the following in a resource lookup:

```
Req: GET /rd-lookup/res?rt=temperature
```

```
Res: 2.05 Content
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
  if="core.s"; anchor="coap://[2001:db8:f1::2]"
```

while a TCP capable client could say:

```
Req: GET /rd-lookup/res?rt=temperature&tt=tcp
```

```
Res: 2.05 Content
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
  if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"
```

Appendix D. Modernized Link Format parsing

The CoRE Link Format as described in [RFC6690] is unsuitable for some use cases of the Resource Directory, and their resolution scheme is often misunderstood by developers familiar with [RFC8288].

For the correct application of base URIs, we describe the interpretation of a Link Format document as a Modernized Link Format. In Modernized Link Format, the document is processed as in Link Format, with the exception of Section 2.1 of [RFC6690]:

- o The URI-reference inside angle brackets ("**<>**") describes the target URI of the link. If it is a relative reference, it is resolved against the base URI of the document.

- o The context of the link is expressed by the "anchor" parameter; if it is a relative reference, it is resolved against the document's base URI. In absence of the "anchor" attribute, the base URI is the link's context.

Content formats derived from [RFC6690] which inherit its resolution rules, like JSON and CBOR link format of [I-D.ietf-core-links-json], can be interpreted in analogy to that.

For where the Resource Directory is concerned, all common forms of links (eg. all the examples of RFC6690) yield identical results. When interpreting data read from ".well-known/core", differences in interpretation only affect links where the absent anchor attribute means "coap://host/" according to RFC6690 and "coap://host/.well-known/core" according to Modernized Link format; those typically only occur in conjunction with the vaguely defined implicit "hosts" relationship.

D.1. For endpoint developers

When developing endpoints, ie. when generating documents that will be submitted to a Resource Directory, the differences between Modernized Link Format and RFC6690 can be ignored as long as all relative references start with a slash, and any of the following applies:

- o There is no anchor attribute, and the context of the link does not matter to the application.

Example: "</sensors>;ct=40"

- o The anchor is a relative reference.

Example: "</t>;anchor="/sensors/temp";rel="alternate"

- o The target is an absolute reference.

Example: "<http://www.example.com/sensors/t123>;anchor="/sensors/temp";rel="describedby"

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

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

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

Peter van der Stok
consultant

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

Christian Amsuess (editor)
Hollandstr. 12/4
1020
Austria

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 19, 2018

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI
May 18, 2018

Sensor Measurement Lists (SenML)
draft-ietf-core-senml-16

Abstract

This specification defines a format for representing simple sensor measurements and device parameters in the Sensor Measurement Lists (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), Extensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use one of these media types in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

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 November 19, 2018.

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. Overview	3
2. Requirements and Design Goals	4
3. Terminology	5
4. SenML Structure and Semantics	6
4.1. Base Fields	6
4.2. Regular Fields	7
4.3. SenML Labels	8
4.4. Extensibility	8
4.5. Records and Their Fields	9
4.5.1. Names	9
4.5.2. Units	9
4.5.3. Time	10
4.5.4. Values	11
4.6. Resolved Records	11
4.7. Associating Meta-data	12
4.8. Sensor Streaming Measurement Lists (SensML)	12
4.9. Configuration and Actuation usage	12
5. JSON Representation (application/senml+json)	13
5.1. Examples	14
5.1.1. Single Datapoint	14
5.1.2. Multiple Datapoints	14
5.1.3. Multiple Measurements	15
5.1.4. Resolved Data	16
5.1.5. Multiple Data Types	17
5.1.6. Collection of Resources	17
5.1.7. Setting an Actuator	17
6. CBOR Representation (application/senml+cbor)	18
7. XML Representation (application/senml+xml)	20
8. EXI Representation (application/senml-exi)	22
9. Fragment Identification Methods	25
9.1. Fragment Identification Examples	25

9.2. Fragment Identification for the XML and EXI Formats . . .	26
10. Usage Considerations	26
11. CDDL	27
12. IANA Considerations	29
12.1. Units Registry	29
12.2. SenML Label Registry	33
12.3. Media Type Registrations	34
12.3.1. senml+json Media Type Registration	35
12.3.2. sensml+json Media Type Registration	36
12.3.3. senml+cbor Media Type Registration	37
12.3.4. sensml+cbor Media Type Registration	38
12.3.5. senml+xml Media Type Registration	39
12.3.6. sensml+xml Media Type Registration	41
12.3.7. senml-exi Media Type Registration	42
12.3.8. sensml-exi Media Type Registration	43
12.4. XML Namespace Registration	44
12.5. CoAP Content-Format Registration	44
13. Security Considerations	45
14. Privacy Considerations	46
15. Acknowledgement	46
16. References	46
16.1. Normative References	46
16.2. Informative References	49
Authors' Addresses	51

1. Overview

Connecting sensors to the Internet is not new, and there have been many protocols designed to facilitate it. This specification defines a format and media types for carrying simple sensor information in a protocol such as HTTP [RFC7230] or CoAP [RFC7252]. The SenML format is designed so that processors with very limited capabilities could easily encode a sensor measurement into the media type, while at the same time a server parsing the data could relatively efficiently collect a large number of sensor measurements. SenML can be used for a variety of data flow models, most notably data feeds pushed from a sensor to a collector, and the web resource model where the sensor is requested as a resource representation (e.g., "GET /sensor/temperature").

There are many types of more complex measurements and measurements that this media type would not be suitable for. SenML strikes a balance between having some information about the sensor carried with the sensor data so that the data is self describing but it also tries to make that a fairly minimal set of auxiliary information for efficiency reason. Other information about the sensor can be discovered by other methods such as using the CoRE Link Format [RFC6690].

SenML is defined by a data model for measurements and simple meta-data about measurements and devices. The data is structured as a single array that contains a series of SenML Records which can each contain fields such as an unique identifier for the sensor, the time the measurement was made, the unit the measurement is in, and the current value of the sensor. Serializations for this data model are defined for JSON [RFC8259], CBOR [RFC7049], XML [W3C.REC-xml-20081126], and Efficient XML Interchange (EXI) [W3C.REC-exi-20140211].

For example, the following shows a measurement from a temperature gauge encoded in the JSON syntax.

```
[
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "v": 23.1 }
]
```

In the example above, the array has a single SenML Record with a measurement for a sensor named "urn:dev:ow:10e2073a01080063" with a current value of 23.1 degrees Celsius.

2. Requirements and Design Goals

The design goal is to be able to send simple sensor measurements in small packets from large numbers of constrained devices. Keeping the total size of payload small makes it easy to use SenML also in constrained networks, e.g., in a 6LoWPAN [RFC4944]. It is always difficult to define what small code is, but there is a desire to be able to implement this in roughly 1 KB of flash on a 8 bit microprocessor. Experience with power meters and other large scale deployments has indicated that the solution needs to support allowing multiple measurements to be batched into a single HTTP or CoAP request. This "batch" upload capability allows the server side to efficiently support a large number of devices. It also conveniently supports batch transfers from proxies and storage devices, even in situations where the sensor itself sends just a single data item at a time. The multiple measurements could be from multiple related sensors or from the same sensor but at different times.

The basic design is an array with a series of measurements. The following example shows two measurements made at different times. The value of a measurement is given by the "v" field, the time of a measurement is in the "t" field, the "n" field has a unique sensor name, and the unit of the measurement is carried in the "u" field.

```
[
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "t": 1.276020076e+09,
    "v": 23.5 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "t": 1.276020091e+09,
    "v": 23.6 }
]
```

To keep the messages small, it does not make sense to repeat the "n" field in each SenML Record so there is a concept of a Base Name which is simply a string that is prepended to the Name field of all elements in that record and any records that follow it. So a more compact form of the example above is the following.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063", "u": "Cel", "t": 1.276020076e+09,
    "v": 23.5 },
  { "u": "Cel", "t": 1.276020091e+09,
    "v": 23.6 }
]
```

In the above example the Base Name is in the "bn" field and the "n" fields in each Record are the empty string so they are omitted.

Some devices have accurate time while others do not so SenML supports absolute and relative times. Time is represented in floating point as seconds. Values greater than or equal to 2^{28} represent an absolute time relative to the Unix epoch. Values less than 2^{28} represent time relative to the current time.

A simple sensor with no absolute wall clock time might take a measurement every second, batch up 60 of them, and then send the batch to a server. It would include the relative time each measurement was made compared to the time the batch was sent in each SenML Record. The server might have accurate NTP time and use the time it received the data, and the relative offset, to replace the times in the SenML with absolute times before saving the SenML information in a document database.

3. 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 document also uses the following terms:

SenML Record: One measurement or configuration instance in time presented using the SenML data model.

SenML Pack: One or more SenML Records in an array structure.

SenML Label: A short name used in SenML Records to denote different SenML fields (e.g., "v" for "value").

SenML Field: A component of a record that associates a value to a SenML Label for this record.

SensML: Sensor Streaming Measurement List (see Section 4.8).

SensML Stream: One or more SenML Records to be processed as a stream.

This document uses the terms "attribute" and "tag" where they occur with the underlying technologies (XML, CBOR [RFC7049], and Link Format [RFC6690]), not for SenML concepts per se. Note that "attribute" has been widely used previously as a synonym for SenML "field", though.

All comparisons of text strings are performed byte-by-byte (and therefore necessarily case-sensitive).

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that the operator "***" stands for exponentiation.

4. SenML Structure and Semantics

Each SenML Pack carries a single array that represents a set of measurements and/or parameters. This array contains a series of SenML Records with several fields described below. There are two kinds of fields: base and regular. Both the base fields and the regular fields can be included in any SenML Record. The base fields apply to the entries in the Record and also to all Records after it up to, but not including, the next Record that has that same base field. All base fields are optional. Regular fields can be included in any SenML Record and apply only to that Record.

4.1. Base Fields

Base Name: This is a string that is prepended to the names found in the entries.

Base Time: A base time that is added to the time found in an entry.

Base Unit: A base unit that is assumed for all entries, unless otherwise indicated. If a record does not contain a Unit value, then the Base Unit is used. Otherwise the value found in the Unit (if any) is used.

Base Value: A base value is added to the value found in an entry, similar to Base Time.

Base Sum: A base sum is added to the sum found in an entry, similar to Base Time.

Version: Version number of media type format. This field is an optional positive integer and defaults to 5 if not present. [RFC Editor: change the default value to 10 when this specification is published as an RFC and remove this note]

4.2. Regular Fields

Name: Name of the sensor or parameter. When appended to the Base Name field, this must result in a globally unique identifier for the resource. The name is optional, if the Base Name is present. If the name is missing, Base Name must uniquely identify the resource. This can be used to represent a large array of measurements from the same sensor without having to repeat its identifier on every measurement.

Unit: Unit for a measurement value. Optional.

Value: Value of the entry. Optional if a Sum value is present, otherwise required. Values are represented using basic data types. This specification defines floating point numbers ("v" field for "Value"), booleans ("vb" for "Boolean Value"), strings ("vs" for "String Value") and binary data ("vd" for "Data Value"). Exactly one value field MUST appear unless there is Sum field in which case it is allowed to have no Value field.

Sum: Integrated sum of the values over time. Optional. This field is in the unit specified in the Unit value multiplied by seconds. For historical reason it is named sum instead of integral.

Time: Time when value was recorded. Optional.

Update Time: Period of time in seconds that represents the maximum time before this sensor will provide an updated reading for a measurement. Optional. This can be used to detect the failure of sensors or communications path from the sensor.

4.3. SenML Labels

Table 1 provides an overview of all SenML fields defined by this document with their respective labels and data types.

Name	Label	CBOR Label	JSON Type	XML Type
Base Name	bn	-2	String	string
Base Time	bt	-3	Number	double
Base Unit	bu	-4	String	string
Base Value	bv	-5	Number	double
Base Sum	bs	-6	Number	double
Version	bver	-1	Number	int
Name	n	0	String	string
Unit	u	1	String	string
Value	v	2	Number	double
String Value	vs	3	String	string
Boolean Value	vb	4	Boolean	boolean
Data Value	vd	8	String (*)	string (*)
Value Sum	s	5	Number	double
Time	t	6	Number	double
Update Time	ut	7	Number	double

Table 1: SenML Labels

(*) Data Value is base64 encoded string with URL safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

For details of the JSON representation see Section 5, for the CBOR Section 6, and for the XML Section 7.

4.4. Extensibility

The SenML format can be extended with further custom fields. Both new base and regular fields are allowed. See Section 12.2 for details. Implementations MUST ignore fields they don't recognize unless that field has a label name that ends with the '_' character in which case an error MUST be generated.

All SenML Records in a Pack MUST have the same version number. This is typically done by adding a Base Version field to only the first Record in the Pack, or by using the default value.

Systems reading one of the objects MUST check for the Version field. If this value is a version number larger than the version which the system understands, the system MUST NOT use this object. This allows

the version number to indicate that the object contains structure or semantics that is different from what is defined in the present document beyond just making use of the extension points provided here. New version numbers can only be defined in an RFC that updates this specification or its successors.

4.5. Records and Their Fields

4.5.1. Names

The Name value is concatenated to the Base Name value to yield the name of the sensor. The resulting concatenated name needs to uniquely identify and differentiate the sensor from all others. The concatenated name **MUST** consist only of characters out of the set "A" to "Z", "a" to "z", "0" to "9", "-", ":", ".", "/", and "_"; furthermore, it **MUST** start with a character out of the set "A" to "Z", "a" to "z", or "0" to "9". This restricted character set was chosen so that concatenated names can be used directly within various URI schemes (including segments of an HTTP path with no special encoding; note that a name that contains "/" characters maps into multiple URI path segments) and can be used directly in many databases and analytic systems. [RFC5952] contains advice on encoding an IPv6 address in a name. See Section 14 for privacy considerations that apply to the use of long-term stable unique identifiers.

Although it is **RECOMMENDED** that concatenated names are represented as URIs [RFC3986] or URNs [RFC8141], the restricted character set specified above puts strict limits on the URI schemes and URN namespaces that can be used. As a result, implementers need to take care in choosing the naming scheme for concatenated names, because such names both need to be unique and need to conform to the restricted character set. One approach is to include a bit string that has guaranteed uniqueness (such as a 1-wire address [AN1796]). Some of the examples within this document use the device URN namespace as specified in [I-D.ietf-core-dev-urn]. UUIDs [RFC4122] are another way to generate a unique name. However, the restricted character set does not allow the use of many URI schemes, such as the 'tag' scheme [RFC4151] and the 'ni' scheme [RFC6920], in names as such. The use of URIs with characters incompatible with this set, and possible mapping rules between the two, are outside of the scope of the present document.

4.5.2. Units

If the Record has no Unit, the Base Unit is used as the Unit. Having no Unit and no Base Unit is allowed; any information that may be

required about units applicable to the value then needs to be provided by the application context.

4.5.3. Time

If either the Base Time or Time value is missing, the missing field is considered to have a value of zero. The Base Time and Time values are added together to get the time of measurement.

Values less than 268,435,456 (2^{28}) represent time relative to the current time. That is, a time of zero indicates that the sensor does not know the absolute time and the measurement was made roughly "now". A negative value indicates seconds in the past from roughly "now". Positive values up to 2^{28} indicate seconds in the future from "now". Positive values can be used, e.g., for actuation use when the desired change should happen in the future but the sender or the receiver does not have accurate time available.

Values greater than or equal to 2^{28} represent an absolute time relative to the Unix epoch (1970-01-01T00:00Z in UTC time) and the time is counted same way as the Portable Operating System Interface (POSIX) "seconds since the epoch" [TIME_T]. Therefore the smallest absolute time value that can be expressed (2^{28}) is 1978-07-04 21:24:16 UTC.

Because time values up to 2^{28} are used for presenting time relative to "now" and Time and Base Time are added together, care must be taken to ensure that the sum does not inadvertently reach 2^{28} (i.e., absolute time) when relative time was intended to be used.

Obviously, "now"-referenced SenML records are only useful within a specific communication context (e.g., based on information on when the SenML pack, or a specific record in a SensML stream, was sent) or together with some other context information that can be used for deriving a meaning of "now"; the expectation for any archival use is that they will be processed into UTC-referenced records before that context would cease to be available. This specification deliberately leaves the accuracy of "now" very vague as it is determined by the overall systems that use SenML. In a system where a sensor without wall-clock time sends a SenML record with a "now"-referenced time over a high speed RS 485 link to an embedded system with accurate time that resolves "now" based on the time of reception, the resulting time uncertainty could be within 1 ms. At the other extreme, a deployment that sends SenML wind speed readings over a LEO satellite link from a mountain valley might have resulting reception time values that are easily a dozen minutes off the actual time of the sensor reading, with the time uncertainty depending on satellite locations and conditions.

4.5.4. Values

If only one of the Base Sum or Sum value is present, the missing field is considered to have a value of zero. The Base Sum and Sum values are added together to get the sum of measurement. If neither the Base Sum or Sum are present, then the measurement does not have a sum value.

If the Base Value or Value is not present, the missing field(s) are considered to have a value of zero. The Base Value and Value are added together to get the value of the measurement.

Representing the statistical characteristics of measurements, such as accuracy, can be very complex. Future specification may add new fields to provide better information about the statistical properties of the measurement.

In summary, the structure of a SenML record is laid out to support a single measurement per record. If multiple data values are measured at the same time (e.g., air pressure and altitude), they are best kept as separate records linked through their Time value; this is even true where one of the data values is more "meta" than others (e.g., describes a condition that influences other measurements at the same time).

4.6. Resolved Records

Sometimes it is useful to be able to refer to a defined normalized format for SenML records. This normalized format tends to get used for big data applications and intermediate forms when converting to other formats. Also, if SenML Records are used outside of a SenML Pack, they need to be resolved first to ensure applicable base values are applied.

A SenML Record is referred to as "resolved" if it does not contain any base values, i.e., labels starting with the character 'b', except for Version fields (see below), and has no relative times. To resolve the Records, the applicable base values of the SenML Pack (if any) are applied to the Record. That is, for the base values in the Record or before the Record in the Pack, name and base name are concatenated, base time is added to the time of the Record, if the Record did not contain Unit the Base Unit is applied to the record, etc. In addition the records need to be in chronological order in the Pack. An example of this is shown in Section 5.1.4.

The Version field MUST NOT be present in resolved records if the SenML version defined in this document is used and MUST be present otherwise in all the resolved SenML Records.

Future specification that defines new base fields need to specify how the field is resolved.

4.7. Associating Meta-data

SenML is designed to carry the minimum dynamic information about measurements, and for efficiency reasons does not carry significant static meta-data about the device, object or sensors. Instead, it is assumed that this meta-data is carried out of band. For web resources using SenML Packs, this meta-data can be made available using the CoRE Link Format [RFC6690]. The most obvious use of this link format is to describe that a resource is available in a SenML format in the first place. The relevant media type indicator is included in the Content-Type (ct=) link attribute (which is defined for the Link Format in Section 7.2.1 of [RFC7252]).

4.8. Sensor Streaming Measurement Lists (SensML)

In some usage scenarios of SenML, the implementations store or transmit SenML in a stream-like fashion, where data is collected over time and continuously added to the object. This mode of operation is optional, but systems or protocols using SenML in this fashion MUST specify that they are doing this. SenML defines separate media types to indicate Sensor Streaming Measurement Lists (SensML) for this usage (see Section 12.3.2). In this situation, the SensML stream can be sent and received in a partial fashion, i.e., a measurement entry can be read as soon as the SenML Record is received and does not have to wait for the full SensML Stream to be complete.

If times relative to "now" (see Section 4.5.3) are used in SenML Records of a SensML stream, their interpretation of "now" is based on the time when the specific Record is sent in the stream.

4.9. Configuration and Actuation usage

SenML can also be used for configuring parameters and controlling actuators. When a SenML Pack is sent (e.g., using a HTTP/CoAP POST or PUT method) and the semantics of the target are such that SenML is interpreted as configuration/actuation, SenML Records are interpreted as a request to change the values of given (sub)resources (given as names) to given values at the given time(s). The semantics of the target resource supporting this usage can be described, e.g., using [I-D.ietf-core-interfaces]. Examples of actuation usage are shown in Section 5.1.7.

5. JSON Representation (application/senml+json)

For the SenML fields shown in Table 2, the SenML labels are used as the JSON object member names within JSON objects representing the JSON SenML Records.

Name	label	Type
Base Name	bn	String
Base Time	bt	Number
Base Unit	bu	String
Base Value	bv	Number
Base Sum	bs	Number
Version	bver	Number
Name	n	String
Unit	u	String
Value	v	Number
String Value	vs	String
Boolean Value	vb	Boolean
Data Value	vd	String
Value Sum	s	Number
Time	t	Number
Update Time	ut	Number

Table 2: JSON SenML Labels

The root JSON value consists of an array with one JSON object for each SenML Record. All the fields in the above table MAY occur in the records with member values of the type specified in the table.

Only the UTF-8 [RFC3629] form of JSON is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC8259]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648], with padding omitted.

Systems receiving measurements MUST be able to process the range of floating point numbers that are representable as an IEEE double precision floating point numbers [IEEE.754.1985]. This allows time values to have better than microsecond precision over the next 100 years. The number of significant digits in any measurement is not relevant, so a reading of 1.1 has exactly the same semantic meaning as 1.10. If the value has an exponent, the "e" MUST be in lower case. In the interest of avoiding unnecessary verbosity and speeding up processing, the mantissa SHOULD be less than 19 characters long and the exponent SHOULD be less than 5 characters long.

5.1. Examples

5.1.1. Single Datapoint

The following shows a temperature reading taken approximately "now" by a 1-wire sensor device that was assigned the unique 1-wire address of 10e2073a01080063:

```
[
  { "n": "urn:dev:ow:10e2073a01080063", "u": "Cel", "v": 23.1 }
]
```

5.1.2. Multiple Datapoints

The following example shows voltage and current now, i.e., at an unspecified time.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063:", "n": "voltage", "u": "V", "v": 120.1 },
  { "n": "current", "u": "A", "v": 1.2 }
]
```

The next example is similar to the above one, but shows current at Tue Jun 8 18:01:16.001 UTC 2010 and at each second for the previous 5 seconds.

```
[
  { "bn": "urn:dev:ow:10e2073a0108006:", "bt": 1.276020076001e+09,
    "bu": "A", "bver": 5,
    "n": "voltage", "u": "V", "v": 120.1 },
  { "n": "current", "t": -5, "v": 1.2 },
  { "n": "current", "t": -4, "v": 1.3 },
  { "n": "current", "t": -3, "v": 1.4 },
  { "n": "current", "t": -2, "v": 1.5 },
  { "n": "current", "t": -1, "v": 1.6 },
  { "n": "current", "v": 1.7 }
]
```

As an example of Sensor Streaming Measurement Lists (SensML), the following stream of measurements may be sent via a long lived HTTP POST from the producer of the stream to its consumer, and each measurement object may be reported at the time it was measured:

```
[
  { "bn": "urn:dev:ow:10e2073a01080063", "bt": 1.320067464e+09,
    "bu": "%RH", "v": 21.2 },
  { "t": 10, "v": 21.3 },
  { "t": 20, "v": 21.4 },
  { "t": 30, "v": 21.4 },
  { "t": 40, "v": 21.5 },
  { "t": 50, "v": 21.5 },
  { "t": 60, "v": 21.5 },
  { "t": 70, "v": 21.6 },
  { "t": 80, "v": 21.7 },
  ...
]
```

5.1.3. Multiple Measurements

The following example shows humidity measurements from a mobile device with a 1-wire address 10e2073a01080063, starting at Mon Oct 31 13:24:24 UTC 2011. The device also provides position data, which is provided in the same measurement or parameter array as separate entries. Note time is used to for correlating data that belongs together, e.g., a measurement and a parameter associated with it. Finally, the device also reports extra data about its battery status at a separate time.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063", "bt": 1.320067464e+09,
    "bu": "%RH", "v": 20 },
  { "u": "lon", "v": 24.30621 },
  { "u": "lat", "v": 60.07965 },
  { "t": 60, "v": 20.3 },
  { "u": "lon", "t": 60, "v": 24.30622 },
  { "u": "lat", "t": 60, "v": 60.07965 },
  { "t": 120, "v": 20.7 },
  { "u": "lon", "t": 120, "v": 24.30623 },
  { "u": "lat", "t": 120, "v": 60.07966 },
  { "u": "%EL", "t": 150, "v": 98 },
  { "t": 180, "v": 21.2 },
  { "u": "lon", "t": 180, "v": 24.30628 },
  { "u": "lat", "t": 180, "v": 60.07967 }
]
```

The size of this example represented in various forms, as well as that form compressed with gzip is given in the following table.

Encoding	Size	Compressed Size
JSON	573	206
XML	649	235
CBOR	254	196
EXI	161	184

Table 3: Size Comparisons

5.1.4. Resolved Data

The following shows the example from the previous section show in resolved format.

```
[
  { "n": "urn:dev:ow:10e2073a01080063", "u": "%RH", "t": 1.320067464e+09,
    "v": 20 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lon", "t": 1.320067464e+09,
    "v": 24.30621 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lat", "t": 1.320067464e+09,
    "v": 60.07965 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "%RH", "t": 1.320067524e+09,
    "v": 20.3 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lon", "t": 1.320067524e+09,
    "v": 24.30622 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lat", "t": 1.320067524e+09,
    "v": 60.07965 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "%RH", "t": 1.320067584e+09,
    "v": 20.7 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lon", "t": 1.320067584e+09,
    "v": 24.30623 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lat", "t": 1.320067584e+09,
    "v": 60.07966 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "%EL", "t": 1.320067614e+09,
    "v": 98 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "%RH", "t": 1.320067644e+09,
    "v": 21.2 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lon", "t": 1.320067644e+09,
    "v": 24.30628 },
  { "n": "urn:dev:ow:10e2073a01080063", "u": "lat", "t": 1.320067644e+09,
    "v": 60.07967 }
]
```

5.1.5. Multiple Data Types

The following example shows a sensor that returns different data types.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063:", "n": "temp", "u": "Cel", "v": 23.1 },
  { "n": "label", "vs": "Machine Room" },
  { "n": "open", "vb": false },
  { "n": "nfv-reader", "vd": "aGkgCg" }
]
```

5.1.6. Collection of Resources

The following example shows the results from a query to one device that aggregates multiple measurements from other devices. The example assumes that a client has fetched information from a device at 2001:db8::2 by performing a GET operation on `http://[2001:db8::2]` at Mon Oct 31 16:27:09 UTC 2011, and has gotten two separate values as a result, a temperature and humidity measurement as well as the results from another device at `http://[2001:db8::1]` that also had a temperature and humidity. Note that the last record would use the Base Name from the 3rd record but the Base Time from the first record.

```
[
  { "bn": "2001:db8::2/", "bt": 1.320078429e+09,
    "n": "temperature", "u": "Cel", "v": 25.2 },
  { "n": "humidity", "u": "%RH", "v": 30 },
  { "bn": "2001:db8::1/", "n": "temperature", "u": "Cel", "v": 12.3 },
  { "n": "humidity", "u": "%RH", "v": 67 }
]
```

5.1.7. Setting an Actuator

The following example show the SenML that could be used to set the current set point of a typical residential thermostat which has a temperature set point, a switch to turn on and off the heat, and a switch to turn on the fan override.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063:" },
  { "n": "temp", "u": "Cel", "v": 23.1 },
  { "n": "heat", "u": "/", "v": 1 },
  { "n": "fan", "u": "/", "v": 0 }
]
```

In the following example two different lights are turned on. It is assumed that the lights are on a network that can guarantee delivery of the messages to the two lights within 15 ms (e.g. a network using 802.1BA [IEEE802.1ba-2011] and 802.1AS [IEEE802.1as-2011] for time synchronization). The controller has set the time of the lights coming on to 20 ms in the future from the current time. This allows both lights to receive the message, wait till that time, then apply the switch command so that both lights come on at the same time.

```
[
  { "bt": 1.320078429e+09, "bu": "/", "n": "2001:db8::3", "v": 1 },
  { "n": "2001:db8::4", "v": 1 }
]
```

The following shows two lights being turned off using a non deterministic network that has a high odds of delivering a message in less than 100 ms and uses NTP for time synchronization. The current time is 1320078429. The user has just turned off a light switch which is turning off two lights. Both lights are dimmed to 50% brightness immediately to give the user instant feedback that something is changing. However given the network, the lights will probably dim at somewhat different times. Then 100 ms in the future, both lights will go off at the same time. The instant but not synchronized dimming gives the user the sensation of quick responses and the timed off 100 ms in the future gives the perception of both lights going off at the same time.

```
[
  { "bt": 1.320078429e+09, "bu": "/", "n": "2001:db8::3", "v": 0.5 },
  { "n": "2001:db8::4", "v": 0.5 },
  { "n": "2001:db8::3", "t": 0.1, "v": 0 },
  { "n": "2001:db8::4", "t": 0.1, "v": 0 }
]
```

6. CBOR Representation (application/senml+cbor)

The CBOR [RFC7049] representation is equivalent to the JSON representation, with the following changes:

- o For JSON Numbers, the CBOR representation can use integers, floating point numbers, or decimal fractions (CBOR Tag 4); however a representation SHOULD be chosen such that when the CBOR value is converted back to an IEEE double precision floating point value, it has exactly the same value as the original Number. For the version number, only an unsigned integer is allowed.

- o Characters in the String Value are encoded using a definite length text string (type 3). Octets in the Data Value are encoded using a definite length byte string (type 2).
- o For compactness, the CBOR representation uses integers for the labels, as defined in Table 4. This table is conclusive, i.e., there is no intention to define any additional integer map keys; any extensions will use string map keys. This allows translators converting between CBOR and JSON representations to convert also all future labels without needing to update implementations. The base values are given negative CBOR labels and others non-negative labels.

Name	Label	CBOR Label
Version	bver	-1
Base Name	bn	-2
Base Time	bt	-3
Base Unit	bu	-4
Base Value	bv	-5
Base Sum	bs	-6
Name	n	0
Unit	u	1
Value	v	2
String Value	vs	3
Boolean Value	vb	4
Value Sum	s	5
Time	t	6
Update Time	ut	7
Data Value	vd	8

Table 4: CBOR representation: integers for map keys

- o For streaming SensML in CBOR representation, the array containing the records SHOULD be a CBOR indefinite length array while for non-streaming SenML, a definite length array MUST be used.

The following example shows a dump of the CBOR example for the same sensor measurement as in Section 5.1.2.

```

0000 87 a7 21 78 1b 75 72 6e 3a 64 65 76 3a 6f 77 3a |...!x.urn:dev:ow:|
0010 31 30 65 32 30 37 33 61 30 31 30 38 30 30 36 3a |10e2073a0108006:|
0020 22 fb 41 d3 03 a1 5b 00 10 62 23 61 41 20 05 00 |".A...[.b#aA ..|
0030 67 76 6f 6c 74 61 67 65 01 61 56 02 fb 40 5e 06 |gvoltage.aV...@^.|
0040 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e 74 06 |ffffff..gcurrent.|
0050 24 02 fb 3f f3 33 33 33 33 33 33 a3 00 67 63 75 |$....?.333333..gcu|
0060 72 72 65 6e 74 06 23 02 fb 3f f4 cc cc cc cc cc |rrent.#...?.....|
0070 cd a3 00 67 63 75 72 72 65 6e 74 06 22 02 fb 3f |...gcurrent."...?|
0080 f6 66 66 66 66 66 66 a3 00 67 63 75 72 72 65 6e |.ffffff..gcurren|
0090 74 06 21 02 f9 3e 00 a3 00 67 63 75 72 72 65 6e |t.!...>...gcurren|
00a0 74 06 20 02 fb 3f f9 99 99 99 99 99 9a a3 00 67 |t. ...?.....g|
00b0 63 75 72 72 65 6e 74 06 00 02 fb 3f fb 33 33 33 |current....?.333|
00c0 33 33 33 |333|
00c3

```

In CBOR diagnostic notation (Section 6 of [RFC7049]), this is:

```

[{-2: "urn:dev:ow:10e2073a0108006:",
  -3: 1276020076.001, -4: "A", -1: 5, 0: "voltage", 1: "V", 2: 120.1},
 {0: "current", 6: -5, 2: 1.2}, {0: "current", 6: -4, 2: 1.3},
 {0: "current", 6: -3, 2: 1.4}, {0: "current", 6: -2, 2: 1.5},
 {0: "current", 6: -1, 2: 1.6}, {0: "current", 6: 0, 2: 1.7}]

```

7. XML Representation (application/senml+xml)

A SenML Pack or Stream can also be represented in XML format as defined in this section.

Only the UTF-8 form of XML is allowed. Characters in the String Value are encoded using the escape sequences defined in [RFC8259]. Octets in the Data Value are base64 encoded with URL safe alphabet as defined in Section 5 of [RFC4648].

The following example shows an XML example for the same sensor measurement as in Section 5.1.2.

```

<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a0108006:" bt="1.276020076001e+09"
    bu="A" bver="5" n="voltage" u="V" v="120.1"></senml>
  <senml n="current" t="-5" v="1.2"></senml>
  <senml n="current" t="-4" v="1.3"></senml>
  <senml n="current" t="-3" v="1.4"></senml>
  <senml n="current" t="-2" v="1.5"></senml>
  <senml n="current" t="-1" v="1.6"></senml>
  <senml n="current" v="1.7"></senml>
</sensml>

```

The SenML Stream is represented as a sensml element that contains a series of senml elements for each SenML Record. The SenML fields are represented as XML attributes. For each field defined in this document, the following table shows the SenML labels, which are used for the XML attribute name, as well as the according restrictions on the XML attribute values ("type") as used in the XML senml elements.

Name	Label	Type
Base Name	bn	string
Base Time	bt	double
Base Unit	bu	string
Base Value	bv	double
Base Sum	bs	double
Base Version	bver	int
Name	n	string
Unit	u	string
Value	v	double
String Value	vs	string
Data Value	vd	string
Boolean Value	vb	boolean
Value Sum	s	double
Time	t	double
Update Time	ut	double

Table 5: XML SenML Labels

The RelaxNG [RNC] schema for the XML is:

```
default namespace = "urn:ietf:params:xml:ns:senml"
namespace rng = "http://relaxng.org/ns/structure/1.0"
```

```
senml = element senml {
  attribute bn { xsd:string }?,
  attribute bt { xsd:double }?,
  attribute bv { xsd:double }?,
  attribute bs { xsd:double }?,
  attribute bu { xsd:string }?,
  attribute bver { xsd:int }?,

  attribute n { xsd:string }?,
  attribute s { xsd:double }?,
  attribute t { xsd:double }?,
  attribute u { xsd:string }?,
  attribute ut { xsd:double }?,

  attribute v { xsd:double }?,
  attribute vb { xsd:boolean }?,
  attribute vs { xsd:string }?,
  attribute vd { xsd:string }?
}

sensml =
  element sensml {
    senml+
  }

start = sensml
```

8. EXI Representation (application/senml-exi)

For efficient transmission of SenML over e.g. a constrained network, Efficient XML Interchange (EXI) can be used. This encodes the XML Schema [W3C.REC-xmlschema-1-20041028] structure of SenML into binary tags and values rather than ASCII text. An EXI representation of SenML SHOULD be made using the strict schema-mode of EXI. This mode however does not allow tag extensions to the schema, and therefore any extensions will be lost in the encoding. For uses where extensions need to be preserved in EXI, the non-strict schema mode of EXI MAY be used.

The EXI header MUST include an "EXI Options", as defined in [W3C.REC-exi-20140211], with an schemaId set to the value of "a" indicating the schema provided in this specification. Future revisions to the schema can change the value of the schemaId to allow for backwards compatibility. When the data will be transported over CoAP or HTTP, an EXI Cookie SHOULD NOT be used as it simply makes

things larger and is redundant to information provided in the Content-Type header.

The following is the XSD Schema to be used for strict schema guided EXI processing. It is generated from the RelaxNG.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="urn:ietf:params:xml:ns:senml"
  xmlns:ns1="urn:ietf:params:xml:ns:senml">
  <xs:element name="senml">
    <xs:complexType>
      <xs:attribute name="bn" type="xs:string" />
      <xs:attribute name="bt" type="xs:double" />
      <xs:attribute name="bv" type="xs:double" />
      <xs:attribute name="bs" type="xs:double" />
      <xs:attribute name="bu" type="xs:string" />
      <xs:attribute name="bver" type="xs:int" />
      <xs:attribute name="n" type="xs:string" />
      <xs:attribute name="s" type="xs:double" />
      <xs:attribute name="t" type="xs:double" />
      <xs:attribute name="u" type="xs:string" />
      <xs:attribute name="ut" type="xs:double" />
      <xs:attribute name="v" type="xs:double" />
      <xs:attribute name="vb" type="xs:boolean" />
      <xs:attribute name="vs" type="xs:string" />
      <xs:attribute name="vd" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="sensml">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="ns1:senml" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following shows a hexdump of the EXI produced from encoding the following XML example. Note this example is the same information as the first example in Section 5.1.2 in JSON format.

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml bn="urn:dev:ow:10e2073a01080063:" n="voltage" u="V"
    v="120.1"></senml>
  <senml n="current" u="A" v="1.2"></senml>
</sensml>
```

Which compresses with EXI to the following displayed in hexdump:

```
0000 a0 30 0d 84 80 f3 ab 93 71 d3 23 2b b1 d3 7b b9 |.0.....q.#+...{.|
0010 d1 89 83 29 91 81 b9 9b 09 81 89 81 c1 81 81 b1 |...)).....|
0020 99 d2 84 bb 37 b6 3a 30 b3 b2 90 1a b1 58 84 c0 |....7.:0.....X..|
0030 33 04 b1 ba b9 39 32 b7 3a 10 1a 09 06 40 38   |3....92.:.....@8|
003f
```

The above example used the bit packed form of EXI but it is also possible to use a byte packed form of EXI which can makes it easier for a simple sensor to produce valid EXI without really implementing EXI. Consider the example of a temperature sensor that produces a value in tenths of degrees Celsius over a range of 0.0 to 55.0. It would produce an XML SenML file such as:

```
<sensml xmlns="urn:ietf:params:xml:ns:senml">
  <senml n="urn:dev:ow:10e2073a01080063" u="Cel" v="23.1"></senml>
</sensml>
```

The compressed form, using the byte alignment option of EXI, for the above XML is the following:

```
0000 a0 00 48 80 6c 20 01 06 1d 75 72 6e 3a 64 65 76 |..H.1 ...urn:dev|
0010 3a 6f 77 3a 31 30 65 32 30 37 33 61 30 31 30 38 |:ow:10e2073a0108|
0020 30 30 36 33 02 05 43 65 6c 01 00 e7 01 01 00 03 |0063..Cel.....|
0030 01                                     |.|
0031
```

A small temperature sensor device that only generates this one EXI file does not really need a full EXI implementation. It can simply hard code the output replacing the 1-wire device ID starting at byte 0x14 and going to byte 0x23 with its device ID, and replacing the value "0xe7 0x01" at location 0x31 and 0x32 with the current temperature. The EXI Specification [W3C.REC-exi-20140211] contains the full information on how floating point numbers are represented, but for the purpose of this sensor, the temperature can be converted to an integer in tenths of degrees (231 in this example). EXI stores 7 bits of the integer in each byte with the top bit set to one if there are further bytes. So the first bytes at is set to low 7 bits of the integer temperature in tenths of degrees plus 0x80. In this example $231 \& 0x7F + 0x80 = 0xE7$. The second byte is set to the integer temperature in tenths of degrees right shifted 7 bits. In this example $231 \gg 7 = 0x01$.

9. Fragment Identification Methods

A SenML Pack typically consists of multiple SenML Records and for some applications it may be useful to be able to refer with a Fragment Identifier to a single record, or a set of records, in a Pack. The fragment identifier is only interpreted by a client and does not impact retrieval of a representation. The SenML Fragment Identification is modeled after CSV Fragment Identifiers [RFC7111].

To select a single SenML Record, the "rec" scheme followed by a single number is used. For the purpose of numbering records, the first record is at position 1. A range of records can be selected by giving the first and the last record number separated by a '-' character. Instead of the second number, the '*' character can be used to indicate the last SenML Record in the Pack. A set of records can also be selected using a comma separated list of record positions or ranges.

(We use the term "selecting a record" for identifying it as part of the fragment, not in the sense of isolating it from the Pack -- the record still needs to be interpreted as part of the Pack, e.g., using the base values defined in earlier records)

9.1. Fragment Identification Examples

The 3rd SenML Record from "coap://example.com/temp" resource can be selected with:

```
coap://example.com/temp#rec=3
```

Records from 3rd to 6th can be selected with:

```
coap://example.com/temp#rec=3-6
```

Records from 19th to the last can be selected with:

```
coap://example.com/temp#rec=19-*
```

The 3rd and 5th record can be selected with:

```
coap://example.com/temp#rec=3,5
```

To select the Records from third to fifth, the 10th record, and all from 19th to the last:

```
coap://example.com/temp#rec=3-5,10,19-*
```

9.2. Fragment Identification for the XML and EXI Formats

In addition to the SenML Fragment Identifiers described above, with the XML and EXI SenML formats also the syntax defined in the XPointer element() Scheme [XPointerElement] of the XPointer Framework [XPointerFramework] can be used. (This is required by [RFC7303] for media types using the "+xml" structured syntax suffix. SenML allows this for the EXI formats as well for consistency.)

Note that fragment identifiers are available to the client side only; they are not provided in transfer protocols such as CoAP or HTTP. Thus, they cannot be used by the server in deciding which media type to send. Where a server has multiple representations available for a resource identified by a URI, it might send a JSON or CBOR representation when the client was directed to use an XML/EXI fragment identifier with this. Clients can prevent running into this problem by explicitly requesting an XML or EXI media type (e.g., using the CoAP Accept option) when XML/EXI-only fragment identifier syntax is in use in the URI.

10. Usage Considerations

The measurements support sending both the current value of a sensor as well as an integrated sum. For many types of measurements, the sum is more useful than the current value. For historical reasons, this field is called "sum" instead of "integral" which would more accurately describe its function. For example, an electrical meter that measures the energy a given computer uses will typically want to measure the cumulative amount of energy used. This is less prone to error than reporting the power each second and trying to have something on the server side sum together all the power measurements. If the network between the sensor and the meter goes down over some period of time, when it comes back up, the cumulative sum helps reflect what happened while the network was down. A meter like this would typically report a measurement with the unit set to watts, but it would put the sum of energy used in the "s" field of the measurement. It might optionally include the current power in the "v" field.

While the benefit of using the integrated sum is fairly clear for measurements like power and energy, it is less obvious for something like temperature. Reporting the sum of the temperature makes it easy to compute averages even when the individual temperature values are not reported frequently enough to compute accurate averages. Implementers are encouraged to report the cumulative sum as well as the raw value of a given sensor.

Applications that use the cumulative sum values need to understand they are very loosely defined by this specification, and depending on the particular sensor implementation may behave in unexpected ways. Applications should be able to deal with the following issues:

1. Many sensors will allow the cumulative sums to "wrap" back to zero after the value gets sufficiently large.
2. Some sensors will reset the cumulative sum back to zero when the device is reset, loses power, or is replaced with a different sensor.
3. Applications cannot make assumptions about when the device started accumulating values into the sum.

Typically applications can make some assumptions about specific sensors that will allow them to deal with these problems. A common assumption is that for sensors whose measurement values are always positive, the sum should never get smaller; so if the sum does get smaller, the application will know that one of the situations listed above has happened.

Despite the name sum, the sum field is not useful for applications that maintain a running count of the number of times that an event happened or keeping track of a counter such as the total number of bytes sent on an interface. Data like that can be sent directly in the value field.

11. CDDL

As a convenient reference, the JSON and CBOR representations can be described with the common CDDL [I-D.ietf-cbor-cddl] specification in Figure 1 (informative).

```

SenML-Pack = [1* record]

record = {
  ? bn => tstr,           ; Base Name
  ? bt => numeric,        ; Base Time
  ? bu => tstr,           ; Base Units
  ? bv => numeric,        ; Base Value
  ? bs => numeric,        ; Base Sum
  ? bver => uint,         ; Base Version
  ? n => tstr,            ; Name
  ? u => tstr,            ; Units
  ? s => numeric,         ; Value Sum
  ? t => numeric,         ; Time
  ? ut => numeric,        ; Update Time
  ? ( v => numeric // ; Numeric Value
    vs => tstr //      ; String Value
    vb => bool //      ; Boolean Value
    vd => binary-value ) ; Data Value
  * key-value-pair
}

; now define the generic versions
key-value-pair = ( label => value )

label = non-b-label / b-label
non-b-label = tstr .regex "[A-Zac-z0-9][_:.A-Za-z0-9]*" / uint
b-label = tstr .regex "b[_:.A-Za-z0-9]+" / nint

value = tstr / binary-value / numeric / bool
numeric = number / decfrac

```

Figure 1: Common CDDL specification for CBOR and JSON SenML

For JSON, we use text labels and base64url-encoded binary data (Figure 2).

```

bver = "bver" n = "n" s = "s"
bn = "bn" u = "u" t = "t"
bt = "bt" v = "v" ut = "ut"
bu = "bu" vs = "vs" vd = "vd"
bv = "bv" vb = "vb"
bs = "bs"

```

```

binary-value = tstr ; base64url encoded

```

Figure 2: JSON-specific CDDL specification for SenML

For CBOR, we use integer labels and native binary data (Figure 3).

```

bver = -1  n  = 0    s  = 5
bn  = -2   u  = 1    t  = 6
bt  = -3   v  = 2    ut = 7
bu  = -4   vs = 3    vd = 8
bv  = -5   vb = 4
bs  = -6

```

```
binary-value = bstr
```

Figure 3: CBOR-specific CDDL specification for SenML

12. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this specification.

IANA will create a new registry for "Sensor Measurement Lists (SenML) Parameters". The sub-registries defined in Section 12.1 and Section 12.2 will be created inside this registry.

12.1. Units Registry

IANA will create a registry of SenML unit symbols. The primary purpose of this registry is to make sure that symbols uniquely map to give type of measurement. Definitions for many of these units can be found in location such as [NIST811] and [BIPM]. Units marked with an asterisk are NOT RECOMMENDED to be produced by new implementations, but are in active use and SHOULD be implemented by consumers that can use the related base units.

Symbol	Description	Type	Reference
m	meter	float	RFC-AAAA
kg	kilogram	float	RFC-AAAA
g	gram*	float	RFC-AAAA
s	second	float	RFC-AAAA
A	ampere	float	RFC-AAAA
K	kelvin	float	RFC-AAAA
cd	candela	float	RFC-AAAA
mol	mole	float	RFC-AAAA
Hz	hertz	float	RFC-AAAA
rad	radian	float	RFC-AAAA
sr	steradian	float	RFC-AAAA
N	newton	float	RFC-AAAA
Pa	pascal	float	RFC-AAAA
J	joule	float	RFC-AAAA
W	watt	float	RFC-AAAA

C	coulomb	float	RFC-AAAA
V	volt	float	RFC-AAAA
F	farad	float	RFC-AAAA
Ohm	ohm	float	RFC-AAAA
S	siemens	float	RFC-AAAA
Wb	weber	float	RFC-AAAA
T	tesla	float	RFC-AAAA
H	henry	float	RFC-AAAA
Cel	degrees Celsius	float	RFC-AAAA
lm	lumen	float	RFC-AAAA
lx	lux	float	RFC-AAAA
Bq	becquerel	float	RFC-AAAA
Gy	gray	float	RFC-AAAA
Sv	sievert	float	RFC-AAAA
kat	katal	float	RFC-AAAA
m2	square meter (area)	float	RFC-AAAA
m3	cubic meter (volume)	float	RFC-AAAA
l	liter (volume)*	float	RFC-AAAA
m/s	meter per second (velocity)	float	RFC-AAAA
m/s2	meter per square second (acceleration)	float	RFC-AAAA
m3/s	cubic meter per second (flow rate)	float	RFC-AAAA
l/s	liter per second (flow rate)*	float	RFC-AAAA
W/m2	watt per square meter (irradiance)	float	RFC-AAAA
cd/m2	candela per square meter (luminance)	float	RFC-AAAA
bit	bit (information content)	float	RFC-AAAA
bit/s	bit per second (data rate)	float	RFC-AAAA
lat	degrees latitude (note 1)	float	RFC-AAAA
lon	degrees longitude (note 1)	float	RFC-AAAA
pH	pH value (acidity; logarithmic quantity)	float	RFC-AAAA
dB	decibel (logarithmic quantity)	float	RFC-AAAA
dBW	decibel relative to 1 W (power level)	float	RFC-AAAA
Bspl	bel (sound pressure level; logarithmic quantity)*	float	RFC-AAAA
count	1 (counter value)	float	RFC-AAAA
/	1 (Ratio e.g., value of a switch, note 2)	float	RFC-AAAA
%	1 (Ratio e.g., value of a switch, note 2)*	float	RFC-AAAA
%RH	Percentage (Relative Humidity)	float	RFC-AAAA
%EL	Percentage (remaining battery energy level)	float	RFC-AAAA
EL	seconds (remaining battery energy level)	float	RFC-AAAA
1/s	1 per second (event rate)	float	RFC-AAAA

1/min	1 per minute (event rate, "rpm")*	float	RFC-AAAA
beat/min	1 per minute (Heart rate in beats per minute)*	float	RFC-AAAA
beats	1 (Cumulative number of heart beats)*	float	RFC-AAAA
S/m	Siemens per meter (conductivity)	float	RFC-AAAA

Table 6

- o Note 1: Assumed to be in WGS84 unless another reference frame is known for the sensor.
- o Note 2: A value of 0.0 indicates the switch is off while 1.0 indicates on and 0.5 would be half on. The preferred name of this unit is "/". For historical reasons, the name "%" is also provided for the same unit - but note that while that name strongly suggests a percentage (0..100) -- it is however NOT a percentage, but the absolute ratio!

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider the following guidelines:

1. There needs to be a real and compelling use for any new unit to be added.
2. Each unit should define the semantic information and be chosen carefully. Implementers need to remember that the same word may be used in different real-life contexts. For example, degrees when measuring latitude have no semantic relation to degrees when measuring temperature; thus two different units are needed.
3. These measurements are produced by computers for consumption by computers. The principle is that conversion has to be easily be done when both reading and writing the media type. The value of a single canonical representation outweighs the convenience of easy human representations or loss of precision in a conversion.
4. Use of SI prefixes such as "k" before the unit is not recommended. Instead one can represent the value using scientific notation such a 1.2e3. The "kg" unit is exception to this rule since it is an SI base unit; the "g" unit is provided for legacy compatibility.
5. For a given type of measurement, there will only be one unit type defined. So for length, meters are defined and other

lengths such as mile, foot, light year are not allowed. For most cases, the SI unit is preferred.

(Note that some amount of judgment will be required here, as even SI itself is not entirely consistent in this respect. For instance, for temperature [ISO-80000-5] defines a quantity, item 5-1 (thermodynamic temperature), and a corresponding unit 5-1.a (Kelvin), and then goes ahead to define another quantity right besides that, item 5-2 ("Celsius temperature"), and the corresponding unit 5-2.a (degree Celsius). The latter quantity is defined such that it gives the thermodynamic temperature as a delta from $T_0 = 273.15$ K. ISO 80000-5 is defining both units side by side, and not really expressing a preference. This level of recognition of the alternative unit degree Celsius is the reason why Celsius temperatures exceptionally seem acceptable in the SenML units list alongside Kelvin.)

6. Symbol names that could be easily confused with existing common units or units combined with prefixes should be avoided. For example, selecting a unit name of "mph" to indicate something that had nothing to do with velocity would be a bad choice, as "mph" is commonly used to mean miles per hour.
7. The following should not be used because they are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.
8. The following units should not be used as they are commonly used to represent other measurements: Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, mph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.
9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.
10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.
11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML Label Registry

IANA will create a new registry for SenML labels. The initial content of the registry is:

Name	Label	CL	JSON Type	XML Type	EI	Reference
Base Name	bn	-2	String	string	a	RFC-AAAA
Base Time	bt	-3	Number	double	a	RFC-AAAA
Base Unit	bu	-4	String	string	a	RFC-AAAA
Base Value	bv	-5	Number	double	a	RFC-AAAA
Base Sum	bs	-6	Number	double	a	RFC-AAAA
Base Version	bver	-1	Number	int	a	RFC-AAAA
Name	n	0	String	string	a	RFC-AAAA
Unit	u	1	String	string	a	RFC-AAAA
Value	v	2	Number	double	a	RFC-AAAA
String Value	vs	3	String	string	a	RFC-AAAA
Boolean Value	vb	4	Boolean	boolean	a	RFC-AAAA
Data Value	vd	8	String	string	a	RFC-AAAA
Value Sum	s	5	Number	double	a	RFC-AAAA
Time	t	6	Number	double	a	RFC-AAAA
Update Time	ut	7	Number	double	a	RFC-AAAA

Table 7: IANA Registry for SenML Labels, CL = CBOR Label, EI = EXI ID

This is the same table as Table 1, with notes removed, and with columns added for the information that is all the same for this initial set of registrations, but will need to be supplied with a different value for new registrations.

All new entries must define the Label Name, Label, and XML Type but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The EI column contains the EXI schemaId value of the first Schema which includes this label or is empty if this label was not intended for use with EXI. The Note field SHOULD contain information about where to find out more information about this label.

The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer and int become a JSON Number. XML boolean and string become a JSON Boolean and String respectively. CBOR represents numeric values with a CBOR type that does not lose any information from the JSON value. EXI uses the XML types.

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review. New entries should not be made that counteract the advice at the end of Section 4.5.4.

All new SenML labels that have "base" semantics (see Section 4.1) MUST start with the character 'b'. Regular labels MUST NOT start with that character. All new SenML labels with Value semantics (see Section 4.2) MUST have "Value" in their (long form) name.

Extensions that add a label that is intended for use with XML need to create a new RelaxNG scheme that includes all the labels in the IANA registry.

Extensions that add a label that is intended for use with EXI need to create a new XSD Schema that includes all the labels in the IANA registry and then allocate a new EXI schemaId value. Moving to the next letter in the alphabet is the suggested way to create the new value for the EXI schemaId. Any labels with previously blank ID values SHOULD be updated in the IANA table to have their ID set to this new schemaId value.

Extensions that are mandatory to understand to correctly process the Pack MUST have a label name that ends with the '_' character.

12.3. Media Type Registrations

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303]. This document registers media types for each serialization format of SenML (JSON, CBOR, XML, and EXI) and also a corresponding set of media types for the streaming use (SensML, see Section 4.8). Clipboard formats are defined for the JSON and XML forms of SenML but not for streams or non-textual formats.

The reason there are both SenML and the streaming SensML formats is that they are not the same data formats and they require separate negotiation to understand if they are supported and which one is being used. The non streaming format is required to have some sort of end of pack syntax which indicates there will be no more records. Many implementations that receive SenML wait for this end of pack marker before processing any of the records. On the other hand, with the streaming formats, it is explicitly not required to wait for this end of pack marker. Many implementations that produce streaming SensML will never send this end of pack marker so implementations that receive streaming SensML can not wait for the end of pack marker before they start processing the records. Given the SenML and

streaming SenML are different data formats, and the requirement for separate negotiation, a media type for each one is needed.

Note to RFC Editor - please remove this paragraph. Note that a request for media type review for senml+json was sent to the media-types@iana.org on Sept 21, 2010. A second request for all the types was sent on October 31, 2016. Please change all instances of RFC-AAAA with the RFC number of this document.

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of 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 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON 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. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senml

Windows Clipboard Name: "JSON Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-json
conforms to public.text

Person & email address to contact for further information: Cullen
Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. sensml+json Media Type Registration

Type name: application

Subtype name: sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of 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 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON 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. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 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. The "bver"

field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-cbor
conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. sensml+cbor Media Type Registration

Type name: application

Subtype name: sensml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 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. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlc

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.5. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlx

Windows Clipboard Name: "XML Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-xml conforms to public.xml

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.6. sensml+xml Media Type Registration

Type name: application

Subtype name: sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlx

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.7. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmle

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-exi conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.8. sensml-exi Media Type Registration

Type name: application

Subtype name: sensml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmle

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. IDs for the JSON, CBOR, and EXI Content-Formats are assigned from the "Expert Review" (0-255) range and for the XML Content-Format from the "IETF Review or IESG Approval" range. The assigned IDs are shown in Table 8.

Media type	Encoding	ID	Reference
application/senml+json	-	TBD:110	RFC-AAAA
application/sensml+json	-	TBD:111	RFC-AAAA
application/senml+cbor	-	TBD:112	RFC-AAAA
application/sensml+cbor	-	TBD:113	RFC-AAAA
application/senml-exi	-	TBD:114	RFC-AAAA
application/sensml-exi	-	TBD:115	RFC-AAAA
application/senml+xml	-	TBD:310	RFC-AAAA
application/sensml+xml	-	TBD:311	RFC-AAAA

Table 8: CoAP Content-Format IDs

13. Security Considerations

Sensor data presented with SenML can contain a wide range of information ranging from information that is very public, such as the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. When SenML is used for configuration or actuation, it can be used to change the state of systems and also impact the physical world, e.g., by turning off a heater or opening a lock.

The SenML formats alone do not provide any security and instead rely on the protocol that carries them to provide security. Applications using SenML need to look at the overall context of how these formats will be used to decide if the security is adequate. In particular for sensitive sensor data and actuation use it is important to ensure that proper security mechanisms are used to provide, e.g., confidentiality, data integrity, and authentication as appropriate for the usage.

The SenML formats defined by this specification do not contain any executable content. However, future extensions could potentially embed application specific executable content in the data.

SenML Records are intended to be interpreted in the context of any applicable base values. If records become separated from the record that establishes the base values, the data will be useless or, worse, wrong. Care needs to be taken in keeping the integrity of a Pack that contains unresolved SenML Records (see Section 4.6).

See also Section 14.

14. Privacy Considerations

Sensor data can range from information with almost no privacy considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transfer protocol such as S/MIME [RFC5751] or HTTP with TLS [RFC2818] that can provide integrity, confidentiality, and authentication information about the source of the data.

The name fields need to uniquely identify the sources or destinations of the values in a SenML Pack. However, the use of long-term stable unique identifiers can be problematic for privacy reasons [RFC6973], depending on the application and the potential of these identifiers to be used in correlation with other information. They should be used with care or avoided as for example described for IPv6 addresses in [RFC7721].

15. Acknowledgement

We would like to thank Alexander Pelov, Alexey Melnikov, Andrew McClure, Andrew McGregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Jim Schaad, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, Peter Saint-Andre, Roni Even, and Stephen Farrell, for their review comments.

16. References

16.1. Normative References

- [BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.
- [IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.
- [NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.

- [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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.
- [RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <<https://www.rfc-editor.org/info/rfc7303>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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>>.

- [RNC] ISO/IEC, "Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG", ISO/IEC 19757-2, Annex C: RELAX NG Compact syntax, December 2008.
- [TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.
- [W3C.REC-exi-20140211] Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-exi-20140211, February 2014, <<http://www.w3.org/TR/2014/REC-exi-20140211>>.
- [W3C.REC-xml-20081126] 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>>.
- [W3C.REC-xmlschema-1-20041028] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>>.
- [XPointerElement] Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer element() Scheme", W3C Recommendation REC-xptr-element, March 2003, <<https://www.w3.org/TR/2003/REC-xptr-element-20030325/>>.
- [XPointerFramework] Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer Framework", W3C Recommendation REC-XPointer-Framework, March 2003, <<http://www.w3.org/TR/2003/REC-xptr-framework-20030325/>>.

16.2. Informative References

- [AN1796] Linke, B., "Overview of 1-Wire Technology and Its Use", June 2008, <<http://pdfserv.maximintegrated.com/en/an/AN1796.pdf>>.
- [I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.
- [I-D.ietf-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-ietf-core-dev-urn-01 (work in progress), March 2018.
- [I-D.ietf-core-interfaces] Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-11 (work in progress), March 2018.
- [IEEE802.1as-2011] IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", 2011.
- [IEEE802.1ba-2011] IEEE, "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems", 2011.
- [ISO-80000-5] "Quantities and units - Part 5: Thermodynamics", ISO 80000-5, Edition 1.0, May 2007.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [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>>.

- [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>>.
- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <<https://www.rfc-editor.org/info/rfc4151>>.
- [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>>.
- [RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <<https://www.rfc-editor.org/info/rfc5751>>.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <<https://www.rfc-editor.org/info/rfc5952>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [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>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7111] Hausenblas, M., Wilde, E., and J. Tennison, "URI Fragment Identifiers for the text/csv Media Type", RFC 7111, DOI 10.17487/RFC7111, January 2014, <<https://www.rfc-editor.org/info/rfc7111>>.
- [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>>.

- [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>>.
- [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>>.
- [UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <<http://unitsofmeasure.org/ucum.html>>.

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
June 04, 2018

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-04

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 December 6, 2018.

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. Security Considerations	11
6. IANA Considerations	11
6.1. "SID mega-range" registry	11
6.1.1. IANA SID Mega-Range Registry	12
6.1.2. IANA "RFC SID range assignment" sub-registries	13
6.2. "YANG module assignment" registry	14
7. Acknowledgments	14
8. References	14
8.1. Normative References	14
8.2. Informative References	15
Appendix A. ".sid" file example	16
Authors' Addresses	25

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. 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 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

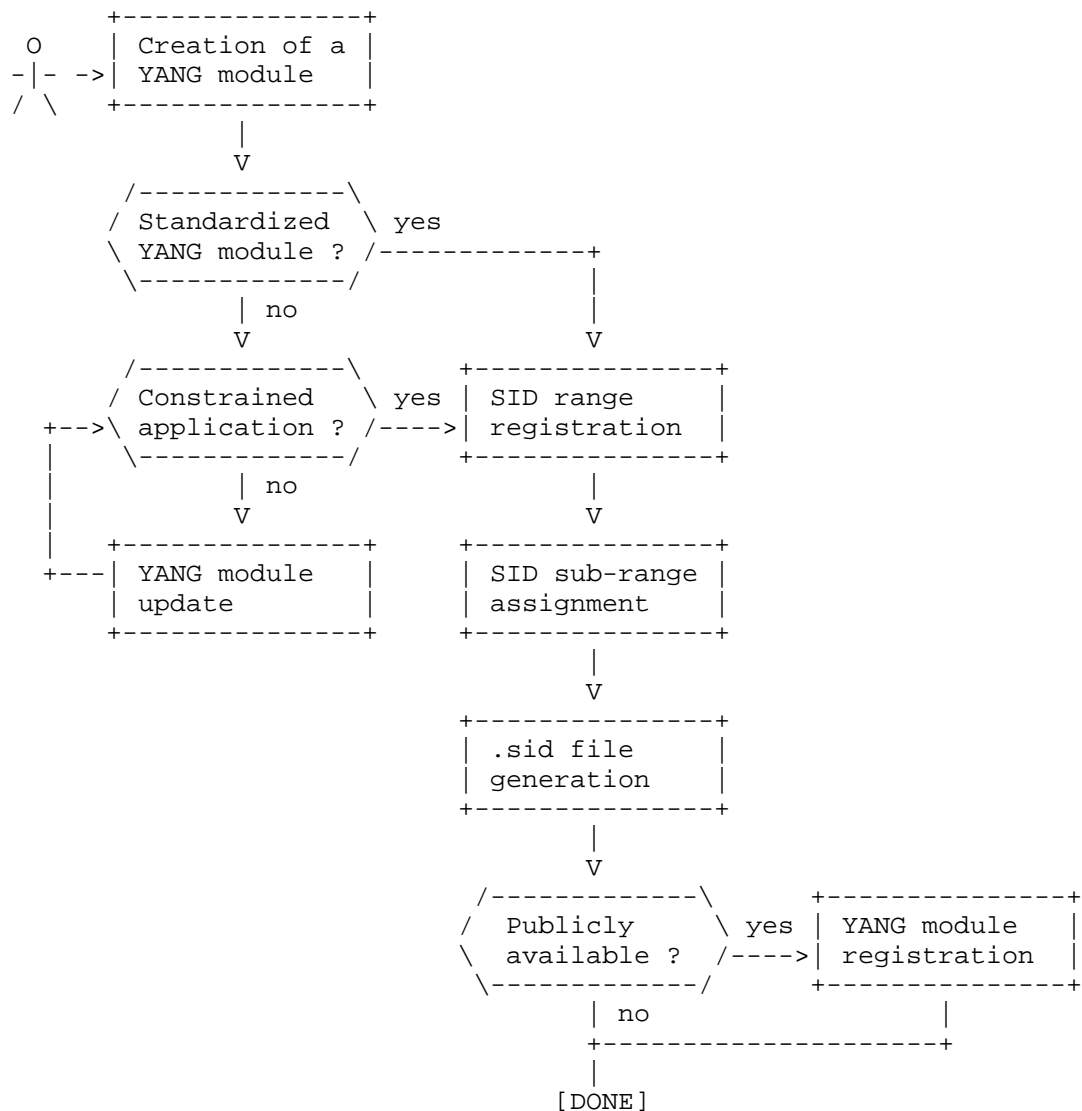
YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoMI [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

YANG modules are not necessary 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. This process starts by the registration of a SID range. Once a SID range is registered, the owner of this range assigns sub-ranges to each YANG module in order to generate the associated ".sid" files. Generation of ".sid" files SHOULD be performed using an automated tool.

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.

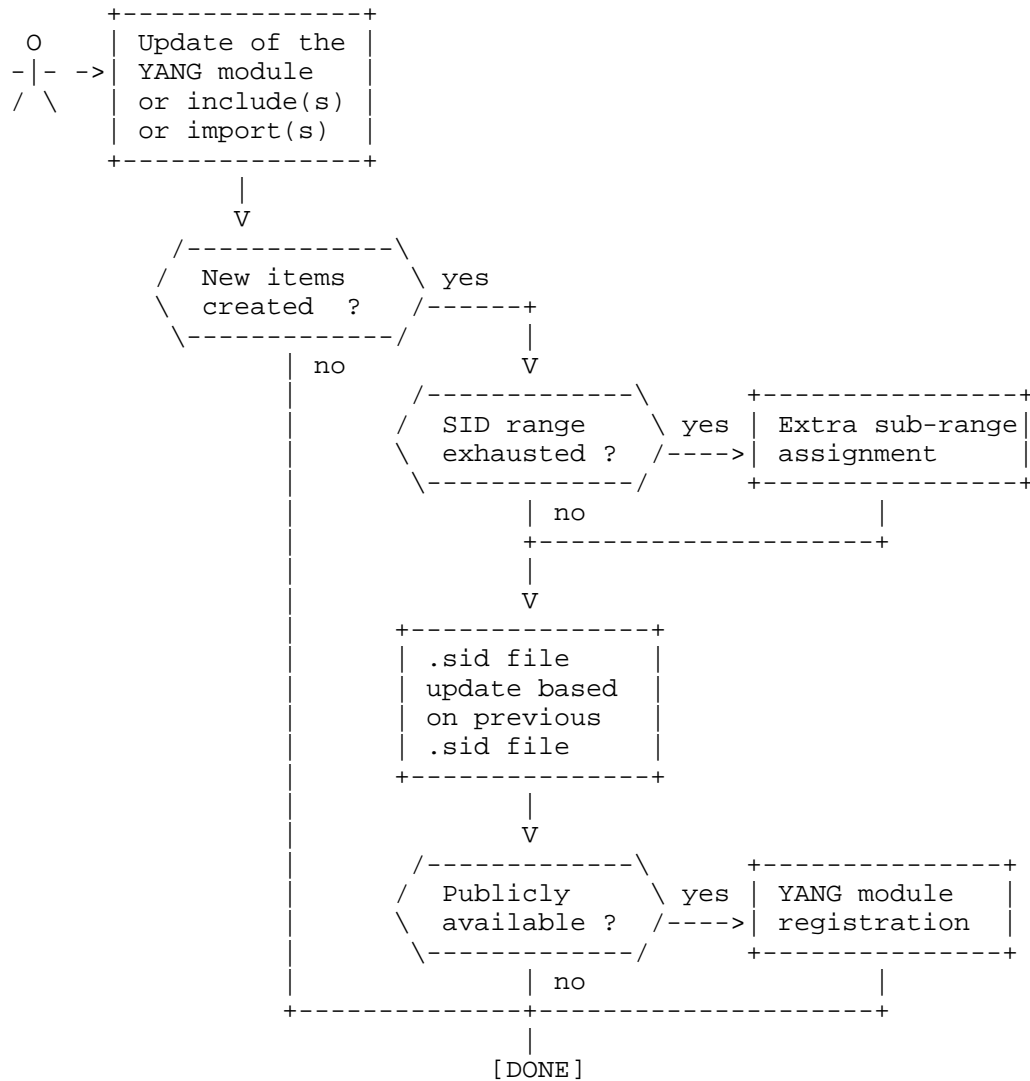
The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



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'. 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. 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.

6. IANA Considerations

6.1. "SID mega-range" registry

The name of this registry is "SID mega-range". This registry is used to delegate the management of block of SIDs for third party's (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:
 - * Organization name
 - * Primary contact name, email address, and phone number
 - * 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).

6.1.1. IANA SID Mega-Range Registry

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].
- 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 prior to the RFC publication but should not be done prior to the working group adoption.
- 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	1,000,000,000	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.

6.1.2. IANA "RFC SID range assignment" sub-registries

The name of this sub-registry is "RFC SID range assignment". This sub-registry corresponds to the SID entry point 1000, size 59000. Each entry in this sub-registry must include the SID range entry point, the SID range size, the YANG module name, 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.

6.2. "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 sub-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 verify.

The IANA policy for future additions to this registry is "First Come First Served" as described in [RFC5226].

7. Acknowledgments

The authors would like to thank Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain and Randy Turner for their help during the development of this document and their useful comments during the review process.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [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>>.

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-02 (work in progress), December 2017.
- [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>>.
- [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

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

Email: a@ackl.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

A. Keranen
Ericsson
July 2, 2018

Too Many Requests Response Code for the Constrained Application Protocol
draft-ietf-core-too-many-reqs-02

Abstract

A Constrained Application Protocol (CoAP) server can experience temporary overload because one or more clients are sending requests to the server at a higher rate than the server is capable or willing to handle. This document defines a new CoAP Response Code for a server to indicate that a client should reduce the rate of 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 <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 January 3, 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. Terminology	3
3. CoAP Server Behavior	3
4. CoAP Client Behavior	3
5. Security Considerations	3
6. IANA Considerations	3
7. Acknowledgements	4
8. References	4
8.1. Normative References	4
8.2. Informative References	4
Author's Address	5

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] Response Codes are used by a CoAP server to indicate the result of the attempt to understand and satisfy a request sent by a client.

CoAP Response Codes are similar to the HTTP [RFC7230] Status Codes and many codes are shared with similar semantics by both CoAP and HTTP. HTTP has the code "429" registered for "Too Many Requests" [RFC6585]. This document registers a CoAP Response Code "4.29" for similar purpose and also defines use of the Max-Age option to indicate a back-off period after which a client can try the request again.

While a server may not be able to response to a one kind of request, it may be able to respond to a different request, even from the same client. Therefore the back-off period applies only to similar requests. For the purpose of this response code, a request is similar if it has the same method, Request-URI, and payload. Also if a client is sending a sequence of requests that are part of the same series (e.g., a set of measurements to be processed by the server) they can be considered similar even if request payloads or URIs may be different. Because request similarity is context-dependent, it is up to the application logic to decide how similar requests should be suppressed.

The 4.29 code is similar to the 5.03 "Service Unavailable" [RFC7252] code in a way that the 5.03 code can also be used by a server to signal an overload situation. However the 4.29 code indicates that the too frequent requests from the requesting client are the reason for the overload.

2. Terminology

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

Readers should also be familiar with the terms and concepts discussed in [RFC7252].

3. CoAP Server Behavior

If a CoAP server is unable to serve a client that is sending CoAP request messages more often than the server is capable or willing to handle, the server SHOULD respond to the request(s) with the Response Code 4.29, "Too Many Requests". The Max-Age option is used to indicate the number of seconds after which the server assumes it is OK for the client to retry the request.

An action result payload (see Section 5.5.1 in [RFC7252]) can be sent by the server to give more guidance to the client, e.g., about the details of the overload situation.

4. CoAP Client Behavior

If a client receives the 4.29 Response Code from a CoAP server to a request, it SHOULD NOT send a similar request to the server before the time indicated in the Max-Age option has passed.

A client MUST NOT rely on a server being able to send the 4.29 Response Code in an overload situation because an overloaded server may not be able to reply to all requests at all.

5. Security Considerations

Replying to CoAP requests with a Response Code consumes resources from a server. For a server under attack it may be more appropriate to simply drop requests without responding.

If a CoAP reply with the Too Many Requests Response Code is not authenticated and integrity protected, an attacker can attempt to spoof a reply and make the client wait for an extended period of time before trying again.

6. IANA Considerations

IANA is requested to register the following Response Code in the "CoRE Parameters Registry", "CoAP Response Codes" sub-registry:

- o Response Code: 4.29
- o Description: Too Many Requests
- o Reference: [[This document]]

7. Acknowledgements

This Response Code definition was originally part of the "Publish-Subscribe Broker for CoAP" document [I-D.ietf-core-coap-pubsub]. Author would like to thank Abhijan Bhattacharyya, Carsten Bormann, Gyorgy Rethy, Klaus Hartke, and Sandor Katona for their contributions and reviews.

8. References

8.1. Normative References

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

8.2. Informative References

- [I-D.ietf-core-coap-pubsub] Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (work in progress), March 2018.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/info/rfc6585>>.
- [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>>.

Author's Address

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 11, 2018

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
Acklio
A. Somaraju
Tridonic GmbH & Co KG
R. Turner
Landis+Gyr
A. Minaburo
Acklio
February 07, 2018

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-06

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 August 11, 2018.

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 Data Node Instances	7
4.1. The 'leaf' Data Node	7
4.2. The 'container' Data Node	7
4.2.1. SIDs as keys	8
4.2.2. Member names as keys	10
4.3. The 'leaf-list' Data Node	10
4.4. The 'list' Data Node	11
4.4.1. SIDs as keys	11
4.4.2. Member names as keys	14
4.5. The 'anydata' Data Node	15
4.6. The 'anyxml' Data Node	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	32
10. References	33
10.1. Normative References	33
10.2. Informative References	33
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	false	false	f4
	7/21	true	true	f5
Null	7/22	null	null	f6
Not assigned	7/23	undefined	undefined	f7

Table 1: CBOR diagnostic notation summary

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.

Basic schema nodes such as leaf, leaf-list, list, anydata and anyxml can be encoded standalone. In this case, only the value of this

schema node is encoded in CBOR. Identification of this value needs to be provided by some external means when required.

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. The end user of this mapping specification (e.g. RESTCONF [RFC8040], CoMI [I-D.ietf-core-comi]) 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 data nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

4. Encoding of YANG Data 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' Data Node

Leafs MUST be encoded based on the encoding rules specified in Section 6.

4.2. The 'container' Data Node

Collections such as containers, list instances, notifications, 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 data node identifier, each value is set to the value of this data node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 2.1 encoded as deltas and member names as defined in [RFC7951] encoded using CBOR text strings. The use of CBOR byte strings for keys is reserved for future extensions.

4.2.1. SIDs as keys

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 value. Keys are represented as the delta of the associated SID, delta values are computed as follows:

- o The delta value is equal to the SID of the current schema node minus the SID of the parent schema node. When no parent exists in the context of use of this container, the delta is set to the SID of the current schema node (i.e., a parent with SID equal to zero is assumed).
- o Delta values may result in a negative number, clients and servers MUST support both unsigned and negative deltas.

The following example shows the encoding of a 'system-state' container instance with a single child, a clock container. The clock container has two children, a 'current-datetime' leaf and a 'boot-datetime' leaf.

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;
    }
  }
}
```

For this first representation, we assume that the SID of the parent container (i.e. 'system-state') is not available to the serializer. In this case, root data nodes are encoded using absolute SIDs.

CBOR diagnostic notation:

```

{
  1717 : {
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00" / boot-datetime (SID 1718) /
  }
}

```

CBOR encoding:

```

a1                                     # map(1)
  19 06b5                             # unsigned(1717)
  a2                                   # map(2)
    02                                # unsigned(2)
    78 1a                             # text(26)
    323031352d31302d30325431343a34373a32345a2d30353a3030
    01                                # unsigned(1)
    78 1a                             # text(26)
    323031352d30392d31355430393a31323a35385a2d30353a3030

```

On the other hand, if the serializer is aware of the parent SID, 1716 in the case 'system-state' container, root data nodes are encoded using deltas.

CBOR diagnostic notation:

```

{
  +1 : {
    +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1719)/
    +1 : "2015-09-15T09:12:58Z-05:00" / boot-datetime (SID 1718) /
  }
}

```

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

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 for all members of a top-level collection, and then also whenever the namespaces of the schema node and its parent are different. 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. This example is described in Section 4.2.1.

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)
    696574662d73797374656d3a636c66636b  # "ietf-system:clock"
  a2                              # map(2)
    70                              # text(16)
      63757272656e7442d6461746574696d65  # "current-datetime"
    78 1a                          # text(26)
      323031352d31302d30325431343a34373a32345a2d30353a3030
    6d                              # text(13)
      626f66f742d6461746574696d65  # "boot-datetime"
    78 1a                          # text(26)
      323031352d30392d31355430393a31323a35385a2d30353a3030
```

4.3. The 'leaf-list' Data Node

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

The following example shows the encoding a 'search' leaf-list instance containing the 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' Data Node

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 same rules as a YANG container as defined in Section 4.2.

4.4.1. SIDs as keys

The following example show the encoding of a 'server' list instance using SIDs. It is important to note that the protocol or method using this mapping may carry a parent SID or may have the knowledge of this parent SID based on its context. In these cases, delta encoding can be performed based on this parent SID which minimizes the size of the encoded data.

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;
  }
}
```

CBOR diagnostic notation:

```
[
  {
    1755 : "NRC TIC server",           / name (SID 1755) /
    1757 : {                           / udp (SID 1757) /
      +1 : "tic.nrc.ca",               / address (SID 1758) /
      +2 : 123                         / port (SID 1759) /
    },
    1753 : 0,                          / association-type (SID 1753) /
    1754 : false,                       / iburst (SID 1754) /
    1756 : true                         / prefer (SID 1756) /
  },
  {
    1755 : "NRC TAC server",           / name (SID 1755) /
    1757 : {                           / udp (SID 1757) /
      +1 : "tac.nrc.ca"                / address (SID 1758) /
    }
  }
]
```

CBOR encoding:

```
82                                     # array(2)
a5                                     # map(5)
  19 06db                             # unsigned(1755)
  6e                                     # text(14)
    4e52432054494320736572766572     # "NRC TIC server"
  19 06dd                             # unsigned(1757)
  a2                                     # map(2)
    01                                 # unsigned(1)
    6a                                 # text(10)
      74696332e6e72632e6361         # "tic.nrc.ca"
    02                                 # unsigned(2)
    18 7b                             # unsigned(123)
  19 06d9                             # unsigned(1753)
  00                                 # unsigned(0)
  19 06da                             # unsigned(1754)
  f4                                 # primitive(20)
  19 06dc                             # unsigned(1756)
  f5                                 # primitive(21)
a2                                     # map(2)
  19 06db                             # unsigned(1755)
  6e                                     # text(14)
    4e52432054414320736572766572     # "NRC TAC server"
  19 06dd                             # unsigned(1757)
  a1                                     # map(1)
    01                                 # unsigned(1)
    6a                                 # text(10)
      74616332e6e72632e6361         # "tac.nrc.ca"
```

4.4.2. Member names as keys

The following example shows the encoding of a 'server' list instance using names. This example is described in Section 4.4.1.

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' Data Node

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 Keys of any inner data 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 Values MUST follow the encoding rules of one of the datatypes listed in Section 6.

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

Definition example:

```
anydata event;
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault { # SID 2600
    leaf port-name {                # SID 2601
      type string;
    }
    leaf port-fault {                # SID 2601
      type string;
    }
  }
}
```

CBOR diagnostic notation:

```
{
  2601 : "0/4/21",          / port-name /
  2602 : "Open pin 2"       / port-fault /
}
```

CBOR encoding:

```
a2                                # map(2)
 19 0a29                          # unsigned(2601)
 66                               # text(6)
   302f342f3231                  # "0/4/21"
 19 0a2a                          # unsigned(2602)
 6a                               # text(10)
   4f70656e2070696e2032         # "Open pin 2"
```

4.6. The 'anyxml' Data Node

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.

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.

The encoding rules defined for YANG containers in section 4.2 may be used to serialize YANG data templates.

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;
    }
  }
}

```

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

5.1. SIDs as keys

This example shows a serialization example of the yang-errors template using SIDs as CBOR map key.

CBOR diagnostic notation:

```

{
  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 : "max value exceeded" / error-message (SID 1027) /
  }
}

```

CBOR encoding:

```

A1                                     # map(1)
19 0400                             # unsigned(1024)
A4                                   # map(4)
    04                              # unsigned(4)
    19 03F3                         # unsigned(1011)
    01                              # unsigned(1)
    19 03FA                         # unsigned(1018)
    02                              # unsigned(2)
    19 06CC                         # unsigned(1740)
    03                              # unsigned(3)
    76                              # text(22)
    6D6178696D756D2076616C75655206578636565646564

```

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" : "max value exceeded"
  }
}
```

CBOR encoding:

```

A1                                     # map(1)
  6F                                   # text(15)
    696574662D636F6D693A6572726F72
A4                                     # map(4)
  69                                   # text(9)
    6572726F722D746167               # "error-tag"
  6D                                   # text(13)
    696E76616C69642D76616C7565
  6D                                   # text(13)
    6572726F722D6170702D746167
  6C                                   # text(12)
    6E6F742D696E2D72616E6765
  6F                                   # text(15)
    6572726F722D646174612D6E6F6465
  73                                   # text(19)
    74696D657A6F6E652D7574632D6F6666736574
  6D                                   # text(13)
    6572726F722D6D657373616765
  72                                   # text(18)
    6D61782076616C7565206578636565646564

```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list data node depends on the built-in type of that data 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 data 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 true (major type 7, additional information 21) or false data item (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 1180).

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: 1180

CBOR encoding: 19 049c

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 another 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

69616e612d696662d747970653a657468657226e657443736d616364

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 more information about 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}([^\:]+\:[^\:]+\:)|(\.*\.*))|((([^\:]+\:)*[^\:]+\:
      ?::([^\:]+\:)*[^\:]+\:)?)(%[\p{N}\p{L}]+)?';
  }
}

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 data node. In the case of a single instance data node, a data node defined at the root of a YANG module or submodule or data nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a data 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 data nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted data node SID.

Data 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 data node SID.
- o The following entries MUST contain the value of each key required to identify the instance of the targeted data node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

First example:

The following example shows the encoding of a leaf instance of type instance-identifier which identifies the data node "/system/contact" (SID 1737).

Definition example from [RFC7317]:

```
container system {  
    leaf contact {  
        type string;  
    }  
  
    leaf hostname {  
        type inet:domain-name;  
    }  
}
```

CBOR diagnostic notation: 1737

CBOR encoding: 19 06c9

Second example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the data node instance

"/system/authentication/user/authorized-key/key-data" (SID 1730) for user name "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: [1730, "bob", "admin"]

CBOR encoding:

83	# array(3)
19 06c2	# unsigned(1730)
63	# text(3)
626f62	# "bob"
65	# text(5)
61646d696e	# "admin"

Third example:

The following example shows the encoding of a leaf instance of type instance-identifier which identify the list instance "/system/authentication/user" (SID 1726) corresponding to the user name "jack".

CBOR diagnostic notation: [1726, "jack"]

CBOR encoding:

```

82                # array(2)
  19 06be         # unsigned(1726)
  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 2f20696574662d73797374656d3a73797374656d2f636f6e74616374
```

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

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

CBOR encoding:

```

78 59
  2f696574662d73797374656d3a73797374656d2f61757468656e74696361
  74696f6e2f757365725b6e616d653d27626f62275d2f617574686f72697a
  65642d6b65795b6e616d653d2761646d696e275d2f6b65792d64617461

```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

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

CBOR encoding:

78 33

```
2f696574662d73797374656d3a73797374656d2f61757468656e74696361
746966f6e2f757365725b6e616d653d27626f62275d
```

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
40	bits	YANG bits datatype	RFC XXXX
41	enumeration	YANG enumeration datatype	RFC XXXX
42	identityref	YANG identityref datatype	RFC XXXX
43	instance-identifier	YANG instance-identifier datatype	RFC XXXX

// RFC Ed.: update Tag values using allocated tags if needed 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-02 (work in progress), December 2017.
- [I-D.ietf-core-sid] Veillette, M. and A. Pelov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-03 (work in progress), December 2017.
- [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: January 4, 2019

I. Jarvinen
M. Kojo
I. Raitahila
University of Helsinki
Z. Cao
Huawei
July 3, 2018

Fast-Slow Retransmission Timeout and Congestion Control Algorithm for
CoAP
draft-jarvinen-core-fasor-00

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 January 4, 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
Authors' Addresses	12

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 [JRCK18].

This document specifies FASOR retransmission timeout and congestion control algorithm. 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 [JRCK18]. 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 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 hypothetical 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.

RTO 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.

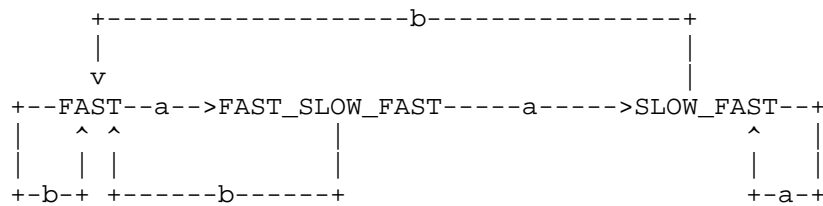
For the initial message, FAST is used with INITIAL_RTO as the FastRTO value. From there on, state is updated when an acknowledgement arrives. Following unambiguous RTT samples, FASOR always uses FAST. Whenever retransmissions are needed, the back off series selection is first downgraded to FAST_SLOW_FAST 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 $2 \times \text{FastRTO}$, 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, $2^1 \times \text{FastRTO}$, ..., $2^i \times \text{FastRTO}$.

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, $\text{Max}(\text{SlowRTO}, 2 \times \text{FastRTO})$, $\text{FastRTO} \times 2^1$, ..., $2^i \times \text{FastRTO}$. 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, $\text{FastRTO} \times 2^1$, ..., $2^i \times \text{FastRTO}$.

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.
- [JRCK18] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "Is
CoAP Congestion Safe?", Applied Networking Research
Workshop (ANRW'18), July 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>>.

Authors' Addresses

Ilpo Jarvinen
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: 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

Iivo Raitahila
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: iivo.raitahila@helsinki.fi

Zhen Cao
Huawei
Beijing
China

EMail: zhencao.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2019

A. Keranen
Ericsson
M. Mohajer
u-blox UK
July 3, 2018

FETCH & PATCH with Sensor Measurement Lists (SenML)
draft-keranen-core-senml-fetch-01

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 semantics for the CoAP iPATCH, 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 <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 January 4, 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. Terminology	3
3. Using FETCH and iPATCH with SenML	3
3.1. SenML FETCH	3
3.2. SenML iPATCH	4
4. Security Considerations	5
5. Acknowledgements	5
6. References	5
6.1. Normative References	5
6.2. Informative References	6
Authors' Addresses	6

1. Introduction

The Sensor Measurement Lists (SenML) media type [I-D.ietf-core-senml] and data model can be used to transmit collections of resources, such as batches of sensor data or configuration parameters.

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 and FETCH methods [RFC8132] enable accessing and updating parts of a resource or multiple resources with one request.

This document defines semantics for the CoAP iPATCH and FETCH methods for resources represented with the SenML data model. Same semantics apply also for the CoAP PATCH 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.

Readers should also be familiar with the terms and concepts discussed in [RFC8132] and [I-D.ietf-core-senml]. 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. Presented using the SenML media type.

PATCH Record: One set of parameters similar to FETCH Record but contains also instructions on how to change existing SenML Pack(s).

PATCH Pack: One or more PATCH Records in an array structure.

3. Using FETCH and iPATCH with SenML

The FETCH and iPATCH methods use the same SenML media type to enable re-use of existing SenML parsers and generators, in particular on constrained devices.

3.1. SenML FETCH

The FETCH method can be used to select and return parts of one or more SenML Packs. The SenML Records are selected by giving the name(s) of the resources using the SenML "name" and/or "base name" Fields.

For example, to select 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 be matched. When time is given in the FETCH Record, only a SenML Record (if any) with equal time value and name is matched.

3.2. SenML iPATCH

The iPATCH 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 method but PATCH Packs can include also new values and other SenML Fields for the Records.

When the name, or time and name, in a PATCH Record matches with an existing Record, the Record is replaced with the contents of the PATCH Record.

If a PATCH Record contains a 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 Records (if any) are removed from the Pack.

For example, the following document could be given as iPATCH 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 iPATCH methods.

Since the the FETCH and iPATCH methods potentially allow retrieving or changing many resources at once, particular care must be taken to ensure that access control rules for different resources are respected.

5. Acknowledgements

The use of FETCH and iPATCH 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 Jaime Jimenez, Klaus Hartke, and also everyone in the IETF CoRE and OMA SpecWorks DMSE working groups for their contributions and reviews.

6. References

6.1. Normative References

- [I-D.ietf-core-senml]
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", draft-ietf-core-senml-16 (work in progress), May 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>>.
- [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>>.

6.2. Informative References

[IPSO] IPSO, "IP for Smart Objects - IPSO Objects", 2018,
<<https://github.com/IPSO-Alliance/pub>>.

Authors' Addresses

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Mojan Mohajer
u-blox UK

Email: Mojan.Mohajer@u-blox.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Silverajan
Tampere University of Technology
T. Savolainen
Nokia Technologies
March 5, 2018

CoAP Communication with Alternative Transports
draft-silverajan-core-coap-alternative-transports-11

Abstract

The aim of this document is to provide a way forward to best decide upon how alternative transport information can be expressed in a CoAP URI. This draft examines the requirements for a new URI format for representing CoAP resources over alternative transports. Various potential URI formats are presented. Benefits and drawbacks of embedding alternative transport information in various ways within the URI components are also discussed. From all listed formats, the document finds scheme-based model to be the most technically feasible.

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 6, 2018.

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. Conformance and Design Considerations	4
3. Situating Transport Information in CoAP URIs	5
3.1. Using the URI scheme component	5
3.1.1. Analysis	6
3.2. Using the URI authority component	6
3.2.1. Analysis	7
3.3. Using the URI path component	7
3.3.1. Analysis	7
3.4. Using the URI query component	7
3.4.1. Analysis	8
4. Discussion	8
5. IANA Considerations	8
6. Security Considerations	9
7. Acknowledgements	9
8. References	9
8.1. Normative References	9
8.2. Informative References	9
Appendix A. Expressing transport in the URI in other ways . . .	10
A.1. Transport information as part of the URI authority . . .	10
A.1.1. Usage of DNS records	11
A.2. Making CoAP Resources Available over Multiple Transports	12
A.3. Transport as part of a 'service:' URL scheme	13
Authors' Addresses	14

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a lightweight, binary application layer protocol designed for constrained environments. Owing to its operating environment, CoAP uses UDP and DTLS as its underlying transports between communicating endpoints. However, with an increase in deployment experiences as well as its popularity, compelling reasons exist for extending CoAP messaging to work over alternative transports. These allow CoAP to better address firewall and NAT traversal issues, to operate in Web browser-based and HTML5 applications as well as for energy-constrained M2M communication in cellular networks. At the time of writing, these transports are:

- o TCP, TLS and Websockets [RFC8323]

- o SMS for cellular networks[I-D.becker-core-coap-sms-gprs]
- o SLIP for serial interfaces[I-D.bormann-t2trg-slipmux]

CoAP uses a REST-based model similar to HTTP, where URIs are used to identify resources at servers. An important factor of allowing CoAP communication over alternative transports, is to express not only the resource identifier, but also the alternative transport information in the URI.

CoAP URIs contain information, such as the endpoint address as well as the location of the resource hosted at the endpoint. CoAP URIs beginning with "coap://" are using UDP, while those beginning with "coaps://" are using DTLS.

```

coap :// server.example.org /sensors/temperature
  \____/      \_____/_____/      \_____/_____/
   |              \              \
URI scheme      URI authority      URI path

```

Figure 1: A CoAP URI

Figure 1 shows the structure of a simple example CoAP URI, in which the various URI components can be interpreted as follows:

- o The URI scheme component (e.g. "coap") contains an application-level identifier which typically identifies the protocol being used as well as its transport and network level protocol configurations. Such configurations are defined by convention or standardisation of the protocol using the scheme.
- o The URI authority component ("server.example.com") contains the endpoint identification, which is typically a fully qualified domain name or a network-level host address.
- o The URI path component ("/sensors/temperature") contains a parameterised resource identifier providing the location and identity of the resource at the endpoint.

In addition to these URI components, Figure 2 shows how specific queries on resource representations are provided by CoAP clients to servers, by specifying one or more URI query components in the URI.

```
coap :// server.example.org /sensors/temperature ?u=cel
                                     \_____/
                                     |
                               URI query
```

Figure 2: A CoAP URI with query

This document focuses on how CoAP URIs can be extended to contain information about alternative transports. For deriving the new URI format, the main design considerations are presented in the next section. Following that, various potential URIs are presented. These URIs provide examples of how transport identifiers can be situated in the URI scheme, authority, path or query components. The proposed URIs are analysed to select feasible formats while disqualifying those not meeting the design criteria.

2. Conformance and Design Considerations

In order to understand which URI formats are best suited for expressing transport information, certain considerations firstly need to be taken into account. Doing so eliminates URI formats that do not meet or conform to the stated requirements. The main criteria are:

1. Conformance to the generic syntax for a URI described in [RFC3986]. A URI format needs to be described in which each URI component clearly meets the syntax and percent-encoding rules described.
2. Alignment with best practices for URI design, as described in [RFC7320]. This is particularly important when it pertains to establishing or standardising the structure and usage of URIs with respect to the various URI components.
3. Request messages sent to a CoAP endpoint using a CoAP Transport URI may be responded to with a relative URI reference. [RFC3986] provides an algorithm to establish how relative references can be resolved against a base URI to obtain a target URI. Given this algorithm, a URI format needs to be described in which relative reference resolution does not result in a target URI that loses its transport-specific information
4. The URI can be supplied as a Proxy-Uri option by a CoAP end-point to a CoAP forward proxy. This allows communication with a CoAP end-point residing in a network using a different transport. Section 6.4 of [RFC7252] provides an algorithm for parsing a received URI to obtain the request's options. Conformance to

[RFC3986] is also necessary in order for the parsing algorithm to be successful.

In addition to the above mentioned requirements, where possible, the following considerations need to be borne in mind:

1. The URI format is able to represent a resource and the transport information for use in constrained environments, without requiring the presence of a naming infrastructure, such as DNS or a directory/lookup service.
 2. Alternative transport information can be easily retrieved by computationally constrained nodes. In other words, the URI format does not result in unnecessarily complex code or logic in such nodes to parse and extract the transport to be used, nor the endpoint address.
 3. URIs are designed to uniquely identify resources. When a single resource is represented with multiple URIs, URI aliasing [WWWArchv1] occurs. Avoiding URI aliasing is considered good practice.
 4. CoAP URIs do not support fragment identifiers.
3. Situating Transport Information in CoAP URIs

The following subsections aim to describe potential URI formats in which the alternative transport information is placed in various URI components.

3.1. Using the URI scheme component

Expressing the transport information in the URI scheme component can be achieved by using new schemes. These can conform to an agreed-upon convention such as "coap+alternative_transport_name" for each new alternative transport and/or "coaps+alternative_transport_name" for its secure counterpart.

Examples of such URIs are:

- o coap+tcp://server.example.org/sensors/temperature for using CoAP over TCP
- o coap+sms://0015105550101/sensors/temperature for using CoAP over SMS with the endpoint identifier being a telephone subscriber number

- o coaps+tcp://server.example.org/sensors/temperature for using CoAP over TLS

3.1.1. Analysis

Expressing transport information in the URI scheme delivers a URI which is human-readable and computationally as easy to parse as standard CoAP URIs, to extract transport identification information. The URI syntax conforms to [RFC3986], and relative URI resolution does not result in the loss of transport identification information. However, each new alternative transport requires minting new schemes, and IANA intervention is required for the registration of each scheme name. The registration process follows the guidelines stipulated in [RFC7595]. Additionally, should a CoAP server wish to expose its resources over multiple transports (such as both UDP and TCP) , URI aliasing can occur if the URI scheme components of these multiple URIs differ in describing the same resource.

3.2. Using the URI authority component

Expressing the transport information within the authority component can result in two possible URI formats.

The first approach is to structure the URI authority's host sub-component with a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- o coap://tcp-server.example.org/sensors/temperature for using CoAP over TCP
- o coap://sms-0015105550101/sensors/temperature for using CoAP over SMS

The second approach is to hint at the alternative transport information, by explicitly specifying using the URI authority's port sub-component, thereby differentiating them from standard CoAP URIs.

Examples of resulting URIs are:

- o coap://server.example.org:5684/sensors/temperature for using CoAP over TLS
- o coap://server.example.org:80/sensors/temperature for using CoAP over WebSockets

3.2.1. Analysis

Embedding the transport information in the host would violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. Consequently, the host in a URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

Embedding the transport information in the port, on the other hand, would not violate the guidelines for the structure of URI authorities in section 2.2 of [RFC7320]. It would result in a CoAP URI that is less human-readable, but URI aliasing is minimised.

On the other hand, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"/server2.example.org/path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI authority component cannot be used as a basis for a new CoAP URI for alternative transports.

3.3. Using the URI path component

Should the URI path component be used, then special characters or keywords need to be supplied in the path to make the transport explicit. Here, many proposals can exist. In general however, this will result in a URI format such as:

- o `coap://server.example.org/sensors/temperature;tcp` for using CoAP over TCP, by appending the transport information at the end of the URI.

3.3.1. Analysis

Embedding the transport information in the URI path directly results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..../path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI path component cannot be used as a basis for a new CoAP URI for alternative transports.

3.4. Using the URI query component

The alternative transport information, should URI query components be used, would result in a URI format such as:

- o `coap://server.example.org/sensors/temperature?alternative-transport=wss` for using CoAP over secure WebSockets.

3.4.1. Analysis

Embedding the transport information in a URI query also results in a URI that is human-readable. However, if a CoAP request message using a CoAP Transport URI of this form elicits a CoAP Response containing a relative URI, for example, of the form `"../..path/to/another/resource"`, relative URI resolution rules of [RFC3986] would result in the loss of transport identification information. Consequently, using the URI query component cannot be used as a basis for a new CoAP URI for alternative transports.

4. Discussion

Based on the analysis of the various options for embedding alternative transport information in a CoAP URI, the most technically feasible option is to use the URI scheme component, as described in Section 3.1. To date, this has also been the WG consensus.

A discussion with IESG members during review of [RFC8323] revealed however, that using the URI scheme to express transport information is not desirable, to avoid the proliferation of new URI schemes for the same application-layer protocol. A strategy was instead proposed to preserve the existing CoAP URI and reuse it for alternative transports, by employing a combination of UDP Confirmable messages and timeouts to determine the eventual correct transport to use between a client and server [IESG-feedback]. The undertaken strategy would have obvious implications regarding interoperability, application and protocol logic, resource usage, for both new CoAP and existing CoAP implementations and deployments. Although URI aliasing can theoretically be avoided with this approach, at the time of writing, its technical feasibility over using the simpler strategy of using URI schemes, has yet to be validated. An obvious drawback is therefore that implementers and other SDOs may choose to provisionally or permanently register new URI schemes with IANA, for CoAP over alternative transports anyway, as was done by the Open Connectivity Foundation (OCF) [CoAP-TCP-TLS-registration].

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

New security risks are not envisaged to arise from the guidelines given in this document, for describing a new URI format containing transport identification within the URI scheme component. However, when specific alternative transports are selected for implementing support for carrying CoAP messages, risk factors or vulnerabilities can be present. Examples include privacy trade-offs when MAC addresses or phone numbers are supplied as URI authority components, or if specific URI path components employed for security-specific interpretations are accidentally encountered as false positives. While this document does not make it mandatory to introduce a security mode with each transport, it recommends ascribing meaning to the use of "coap+" and "coaps+" prefixes in the scheme component, with the "coaps+" prefix used for secure transports for CoAP messages.

7. Acknowledgements

Email discussions, comments and ideas from Thomas Fossati, Akbar Rahman, Klaus Hartke, Martin Thomson, Mark Nottingham, Dave Thaler, Graham Klyne, Carsten Bormann and Markus Becker greatly helped previous versions of this draft.

8. References

8.1. Normative References

- [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>>.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.

8.2. Informative References

- [CoAP-TCP-TLS-registration]
, <<https://www.iana.org>>.

- [I-D.becker-core-coap-sms-gprs]
Kuladinithi, K., Becker, M., Li, K., and T. Poetsch,
"Transport of CoAP over SMS", draft-becker-core-coap-sms-
gprs-06 (work in progress), February 2017.
- [I-D.bormann-t2trg-slipmux]
Bormann, C. and T. Kaupat, "Slipmux: Using an UART
interface for diagnostics, configuration, and packet
transfer", draft-bormann-t2trg-slipmux-02 (work in
progress), January 2018.
- [IESG-feedback]
, <[https://www.ietf.org/mail-archive/web/core/current/
msg08768](https://www.ietf.org/mail-archive/web/core/current/msg08768)>.
- [RFC2609] Guttman, E., Perkins, C., and J. Kempf, "Service Templates
and Service: Schemes", RFC 2609, DOI 10.17487/RFC2609,
June 1999, <<https://www.rfc-editor.org/info/rfc2609>>.
- [RFC7595] 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>>.
- [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>>.
- [WWWArchv1]
, <<http://www.w3>>.

Appendix A. Expressing transport in the URI in other ways

Other means of indicating the transport as a distinguishable component within the CoAP URI are possible, but have been deemed unsuitable by not meeting the design considerations listed, or are incompatible with existing practices outlined in [RFC7252]. They are however, retained in this section for historical documentation and completeness.

A.1. Transport information as part of the URI authority

A single URI scheme, "coap-at" can be introduced, as part of an absolute URI which expresses the transport information within the authority component. One approach is to structure the component with

a transport prefix to the endpoint identifier and a delimiter, such as "<transport-name>-endpoint_identifier".

Examples of resulting URIs are:

- ```
o coap-at://tcp-server.example.com/sensors/temperature
o coap-at://sms-0015105550101/sensors/temperature
```

An implementation note here is that some generic URI parsers will fail when encountering a URI such as "coap-at://tcp-[2001:db8::1]/sensors/temperature". Consequently, an equivalent, but parseable URI from the ip6.arpa domain needs to be formulated instead. For [2001:db8::1] using TCP, this would result in the following URL:

```
coap-at://tcp-1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0
.1.0.0.2.ip6.arpa:5683/sensors/temperature
```

Usage of an IPv4-mapped IPv6 address such as `::ffff.192.100.0.1` can similarly be expressed with a URI from the `ip6.arpa` domain.

This URI format allows the usage of a single scheme to represent multiple types of transport end-points. Consequently, it requires consistency in ensuring how various transport-specific endpoints are identified, as a single URI format is used. Attention must be paid towards the syntax rules and encoding for the URI host component. Additionally, against a base URI of the form "coap-at://tcp-server.example.com/sensors/temperature", resolving a relative reference, such as "//example.net/sensors/temperature" would result in the target URI "coap-at://example.net/sensors/temperature", in which transport information is lost.

### A.1.1. Usage of DNS records

DNS names can be used instead of IPv6 address literals to mitigate lengthy URLs referring to the ip6.arpa domain, if usage of DNS is possible.

DNS SRV records can also be employed to formulate a URL such as:

```
coap-at://srv-coap.tcp.example.com/sensors/temperature
```

in which the "srv" prefix is used to indicate that a DNS SRV lookup should be used for `_coap._tcp.example.com`, where usage of CoAP over TCP is specified for `example.com`, and is eventually resolved to a numerical IPv4 or IPv6 address.

## A.2. Making CoAP Resources Available over Multiple Transports

The CoAP URI used thus far is as follows:

```
URI = scheme ":" hier-part ["?" query]
hier-part = "//" authority path-abempty
```

A new URI format could be introduced, that does not possess an "authority" component, and instead defining "hier-part" to instead use another component, "path-rootless", as specified by RFC3986 [RFC3986]. The partial ABNF format of this URI would then be:

```
URI = scheme ":" hier-part ["?" query]
hier-part = path-rootless
path-rootless = segment-nz *("/" segment)
```

The full syntax of "path-rootless" is described in [RFC3986]. A generic URI defined this way would conform to the syntax of [RFC3986], while the path component can be treated as an opaque string to indicate transport types, endpoints as well as paths to CoAP resources. A single scheme can similarly be used.

A constrained node that is capable of communicating over several types of transports (such as UDP, TCP and SMS) would be able to convey a single CoAP resource over multiple transports. This is also beneficial for nodes performing caching and proxying from one type of transport to another.

Requesting and retrieving the same CoAP resource representation over multiple transports could be rendered possible by prefixing the transport type and endpoint identifier information to the CoAP URI. This would result in the following example representation:

```
coap-at:tcp://example.com?coap://example.com/sensors/temperature
 _____/ _____/
 \ \
 Transport-specific CoAP Resource
 Prefix
```

Figure 3: Prefixing a CoAP URI with TCP transport

Such a representation would result in the URI being decomposed into its constituent components, with the CoAP resource residing within the query component as follows:

Scheme: coap-at

Path: tcp://example.com

Query: coap://example.com/sensors/temperature

The same CoAP resource, if requested over a WebSocket transport, would result the following URI:

coap-at:ws://example.com/endpoint?coap://example.com/sensors/temperature

Transport-specific Prefix                      CoAP Resource

Figure 4: Prefixing a CoAP URI with WebSocket transport

While the transport prefix changes, the CoAP resource representation remains the same in the query component:

Scheme: coap-at

Path: ws://example.com/endpoint

Query: coap://example.com/sensors/temperature

The URI format described here overcomes URI aliasing [WWWArchv1] when multiple transports are used, by ensuring each CoAP resource representation remains the same, but is prefixed with different transports. However, against a base URI of this format, resolving relative references of the form `"/example.net/sensors/temperature"` and `"/sensor2/temperature"` would again result in target URIs which lose transport-specific information.

Implementation note: While square brackets are disallowed within the path component, the '[' and ']' characters needed to enclose a literal IPv6 address can be percent-encoded into their respective equivalents. The ':' character does not need to be percent-encoded. This results in a significantly simpler URI string compared to section 2.2, particularly for compressed IPv6 addresses. Additionally, the URI format can be used to specify other similar address families and formats, such as Bluetooth addresses.

#### A.3. Transport as part of a 'service:' URL scheme

The "service:" URL scheme name was introduced in [RFC2609] and forms the basis of service description used primarily by the Service Location Protocol. An abstract service type URI would have the form `"service:<abstract-type>:<concrete-type>"`

where `<abstract-type>` refers to a service type name that can be associated with a variety of protocols, while the `<concrete-type>`

then providing the specific details of the protocol used, authority and other URI components.

Adopting the "service:" URL scheme to describe CoAP usage over alternative transports would be rather trivial. To use a previous example, a CoAP service to discover a Resource Directory and its base RD resource using TCP would take the form

```
service:coap:tcp://host.example.com/.well-known/core?rt=core-rd
```

The syntax of the "service:" URL scheme differs from the generic URI syntax and therefore such a representation should be treated as an opaque URI as Section 2.1 of [RFC2609] recommends.

#### Authors' Addresses

Bilhanan Silverajan  
Tampere University of Technology  
Korkeakoulunkatu 10  
FI-33720 Tampere  
Finland

Email: bilhanan.silverajan@tut.fi

Teemu Savolainen  
Nokia Technologies  
Hatanpaaen valtatie 30  
FI-33100 Tampere  
Finland

Email: teemu.savolainen@nokia.com

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 3, 2019

B. Silverajan  
TUT  
M. Ocak  
Ericsson  
July 2, 2018

CoAP Protocol Negotiation  
draft-silverajan-core-coap-protocol-negotiation-09

Abstract

CoAP has been standardised as an application-level REST-based protocol. When multiple transport protocols exist for exchanging CoAP resource representations, this document introduces a way forward for CoAP endpoints as well as intermediaries to agree upon alternate transport and protocol configurations as well as URIs for CoAP messaging. Several mechanisms are proposed: Extending the CoRE Resource Directory with new parameter types, introducing a new CoAP Option with which clients can interact directly with servers without needing the Resource Directory, and finally a new CoRE Link Attribute allowing exposing alternate locations on a per-resource basis.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 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. Aim . . . . .                                              | 4  |
| 2.1. Overcoming Middlebox Issues . . . . .                    | 4  |
| 2.2. Better resource caching and serving in proxies . . . . . | 5  |
| 2.3. Interaction with Energy-constrained Servers . . . . .    | 6  |
| 3. Node Types based on Transport Availability . . . . .       | 7  |
| 4. New Resource Directory Parameters . . . . .                | 8  |
| 4.1. The 'at' RD parameter . . . . .                          | 8  |
| 4.2. The 'tt' RD parameter . . . . .                          | 10 |
| 5. CoAP Alternative-Transport Option . . . . .                | 11 |
| 6. The 'ol' CoRE Link Attribute . . . . .                     | 14 |
| 6.1. Using /.well-known/core . . . . .                        | 14 |
| 6.2. Using CoRE Resource Directory . . . . .                  | 15 |
| 7. IANA Considerations . . . . .                              | 15 |
| 8. Security Considerations . . . . .                          | 15 |
| 9. Acknowledgements . . . . .                                 | 16 |
| 10. References . . . . .                                      | 16 |
| 10.1. Normative References . . . . .                          | 16 |
| 10.2. Informative References . . . . .                        | 16 |
| Appendix A. Change Log . . . . .                              | 17 |
| A.1. From -08 to -09 . . . . .                                | 17 |
| A.2. From -07 to -08 . . . . .                                | 17 |
| A.3. From -06 to -07 . . . . .                                | 17 |
| A.4. From -05 to -06 . . . . .                                | 17 |
| A.5. From -04 to -05 . . . . .                                | 17 |
| A.6. From -03 to -04 . . . . .                                | 17 |
| A.7. From -02 to -03 . . . . .                                | 17 |
| A.8. From -01 to -02 . . . . .                                | 18 |
| A.9. From -00 to -01 . . . . .                                | 18 |
| Authors' Addresses . . . . .                                  | 18 |

## 1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] allows clients, origin servers and proxies, to exchange and manipulate resource representations using REST-based methods over UDP or DTLS. CoAP messaging however can use other alternative underlying transports [I-D.silverajan-core-coap-alternative-transports].

When CoAP-based endpoints and proxies possess the ability to perform CoAP messaging over multiple transports, significant benefits can be obtained if communicating client endpoints can discover that multiple transport bindings may exist on an origin server over which CoAP resources can be retrieved. This allows a client to understand and possibly substitute a different transport protocol configuration for the same CoAP resources on the origin server, based on the preferences of the communicating peers. Inevitably, if two CoAP endpoints reside in distinctly separate networks with orthogonal transports, a CoAP proxy node is needed between the two networks so that CoAP Requests and Responses can be exchanged properly.

A URI in CoAP, however, serves two purposes simultaneously. It firstly functions as a locator, by specifying the network location of the endpoint hosting the resource, and the underlying transport used by CoAP for accessing the resource representation. It secondly identifies the name of the specific resource found at that endpoint together with its namespace, or resource path. A single CoAP URI cannot be used to express the identity of the resource independently of alternate underlying transports or protocol configuration. Multiple URIs can result for a single CoAP resource representations if:

- o the authority components of the URI differ, owing to the same physical host exposing several network endpoints. For example, "coap://example.org/sensors/temperature" and "coap://example.net/sensors/temperature"
- o the scheme components of the URI differ, owing to the origin server exposing several underlying transport alternatives. For example, "coap://example.org/sensors/temperature" and "coap+tcp://example.org/sensors/temperature"

Without a priori knowledge, clients would be unable to ascertain if two or more URIs provided by an origin server are associated to the same representation or not. Consequently, a communication mechanism needs to be conceived to allow an origin server to properly capture the relationship between these alternate representations or locations and then subsequently supply this information to clients. This also goes some way in limiting URI aliasing [WWWArchv1].

In order to support CoAP clients, proxies and servers wishing to use CoAP over multiple transports, this draft proposes the following:

- o An ability for servers to register supported CoAP transports to a CoRE Resource Directory [I-D.ietf-core-resource-directory] with optional registration lifetime values

- o A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources
- o New Resource Directory parameter types enabling the above-mentioned features.
- o A new CoAP Option called Alternative-Transport that can be used by CoAP clients to discover and retrieve the types of alternative transports available at the origin server, as well as the links describing the transport-specific endpoint address at which CoAP resources are exposed from.
- o A new CoRE Link attribute for exposing transports and endpoint locations on an origin server on a per-resource basis.

## 2. Aim

The following simple scenarios aim to better portray how CoAP protocol negotiation benefits communicating nodes

### 2.1. Overcoming Middlebox Issues

Discovering which transports are available is important for a client to determine the optimal alternative to perform CoAP messaging according to its needs, particularly when separated from a CoAP server via a NAT. It is well-known that some firewalls as well as many NATs, particularly home gateways, hinder the proper operation of UDP traffic. NAT bindings for UDP-based traffic do not have as long timeouts as TCP-based traffic.

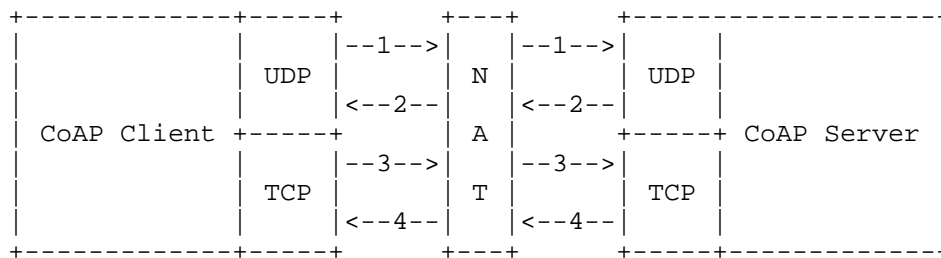


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

Figure 1 depicts such a scenario, where a CoAP client residing behind a NAT uses UDP initially for accessing a CoAP Server, and engages in discovering alternative transports offered by the server. The client subsequently decides to use TCP for CoAP messaging instead of UDP to set up an Observe relationship for a resource at the CoAP Server, in order to avoid incoming packets containing resource updates being discarded by the NAT.

## 2.2. Better resource caching and serving in proxies

Figure 2 outlines a more complex example of intermediate nodes such as CoAP-based proxies to intelligently cache and respond to CoAP or HTTP clients with the same resource representation requested over alternative transports or server endpoints. As with the earlier example, the CoAP Server registers its transports to a Resource Directory (This is assumed to be performed beforehand and not depicted in the figure, for brevity)

In this example, a CoAP over WebSockets client successfully obtains a response from a CoAP forward proxy to retrieve a resource representation from an origin server using UDP, by supplying the CoAP server's endpoint address and resource in a Proxy-URI option. Arrow 1 represents a GET request to "coap+ws://proxy.example.com" which subsequently retrieves the resource from the CoAP server using the URI "coap://example.org/sensors/temperature", shown as arrow 2.

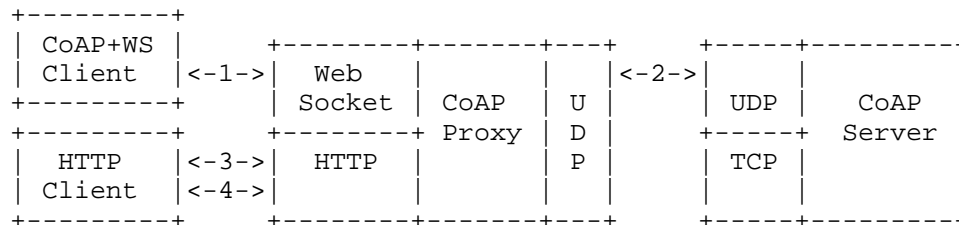


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

Subsequently, assume an HTTP client requests the same resource, but instead specifies a CoAP over TCP alternative URI instead. Arrow 3 represents this event, where the HTTP client performs a GET request to "http://proxy.example.com/coap+tcp://example.org/sensors/temperature". When the proxy receives the request, instead of immediately retrieving the temperature resource again over TCP, it

first verifies either from the Resource Directory or directly from the server, whether the cached resource retrieved over UDP is a valid equivalent representation of the resource requested by the HTTP client over TCP. Upon confirmation, the proxy is able to supply the same cached representation to the HTTP client as well (arrow 4).

### 2.3. Interaction with Energy-constrained Servers

Figure 3 illustrates discovery and communication between a CoAP client and an energy-constrained CoAP Server. Such a server aims at conserving its energy unless a need arises otherwise. The figure first depicts the server registering itself to a Resource Directory over IP, and also supplies its alternative CoAP transport endpoints (in this case, SMS), in steps 1 and 2. The server can subsequently disable communication radio interfaces requiring greater energy (such as for IP-based communication), powering it up sporadically for maintenance activities like registration renewals. At other times, it maintains communication in a low-power state by listening only for incoming SMS messages.

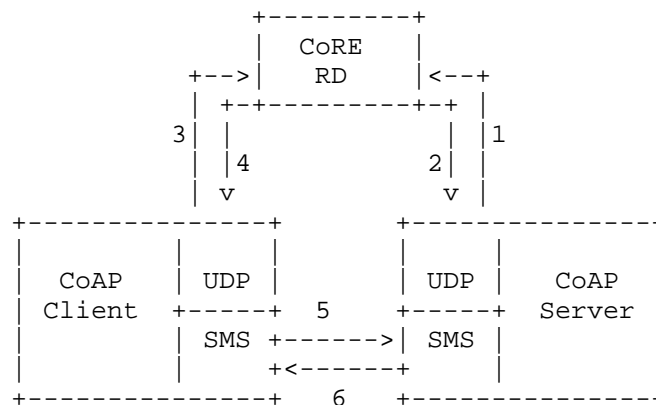


Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

A CoAP client wishing to perform CoAP operations with an energy-constrained CoAP server may query a resource directory for the SMS-based endpoint of the server (steps 3 and 4). Subsequently, SMS-based CoAP communication can occur between the endpoints as shown by arrows 5 and 6. Alternatively, the incoming SMS can be also used by the server as a triggering event to temporarily power up its radio

interface so that UDP or other transport-based CoAP communication can instead be employed for low latency communication with the client.

### 3. Node Types based on Transport Availability

In [RFC7228], Tables 1, 3 and 4 introduced classification schemes for devices, in terms of their resource constraints, energy limitations and communication power. For this document, in addition to these capabilities, it seems useful to also identify devices based on their transport capabilities.

| Name | Transport Availability                                            |
|------|-------------------------------------------------------------------|
| T0   | Single transport                                                  |
| T1   | Multiple transports, with one or more active at any point in time |
| T2   | Multiple active and persistent transports at all times            |

Table 1: Classes of Available Transports

Type T0 nodes possess the capability of exactly 1 type of transport channel for CoAP, at all times. These include both active and sleepy nodes, which may choose to perform duty cycling for power saving.

Type T1 nodes possess multiple different transports, and can retrieve or expose CoAP resources over any or all of these transports. However, not all transports are constantly active and certain transport channels and interfaces could be kept in a mostly-off state for energy-efficiency, such as when using CoAP over SMS.

Type T2 nodes possess more than 1 transport, and multiple transports are simultaneously active at all times in a persistent manner. CoAP proxy nodes which allow CoAP endpoints from disparate transports to communicate with each other, are a good example of this.

#### 4. New Resource Directory Parameters

In order to allow resource interactions between clients and servers with multiple locations or transports, the registration, update and lookup interfaces of the CoRE Resource Directory need to be extended. In this section two new RD parameters, "at" and "tt" are introduced. Both are optional CoAP features. If supported, they occur at the granularity level of an origin server, ie. they cannot be applied selectively on some resources only. When absent, it is assumed that the server does not support multiple transports or locations.

##### 4.1. The 'at' RD parameter

A CoAP server wishing to advertise its resources over multiple transports does so by using one or more "at" parameters to register CoAP alternative transport URIs with a Resource Directory. Such a URI would contain the scheme, address as well as any port or paths at which the server is available.

| Name               | Query | Validity | Description                                     | Value      |
|--------------------|-------|----------|-------------------------------------------------|------------|
| CoAP Transport URI | at    | URI      | URI scheme, address port and path on the server | xsd:string |

Table 2: The "at" RD parameter

The "at" parameter extends the Resource Directory's Registration and Update interfaces.

The following example shows a type T1 endpoint registering its resources and advertising its ability to use TCP and WebSockets as alternative transports:

```
Req: POST coap://rd.example.com/rd?ep=node1
 &at=coap+tcp://[2001:db8:f1::2]&at=coap+ws://server.example.com
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"

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

An endpoint lookup would just reflect the registered attributes:

Req: GET /rd-lookup/ep

Res: 2.05 Content

```
</rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";
 at="coap+tcp://[2001:db8:f1::2]";at="coap+ws://server.example.com"
```

The next example shows the same endpoint updating its registration with a new lifetime and the availability of a single alternative transport for CoAP (in this case TCP):

```
Req: POST /rd/1234?lt=600
 &at=coap+tcp://[2001:db8:f1::2]
Content-Format: 40
Payload:
</temperature>;ct=0;rt="temperature";if="core.s"
```

Res: 2.04 Changed

If a lookup is performed on the same endpoint only 1 alternative transport is indicated:

Req: GET /rd-lookup/ep

Res: 2.05 Content

```
</rd/1234>;ep="node1";base="coap://[2001:db8:f1::2]:5683";
 at="coap+tcp://[2001:db8:f1::2]"
```

A resource lookup for UDP client would be returned as the following:

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content

```
<coap://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
 anchor="coap://[2001:db8:f1::2]"
```

A resource lookup for TCP client would be returned as the following:

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content

```
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";if="core.s";
 anchor="coap+tcp://[2001:db8:f1::2]"
```

#### 4.2. The 'tt' RD parameter

A CoAP client wishing to perform a look-up on the Resource Directory for CoAP servers supporting multiple transports does so by using one or more "tt" parameters to query for CoAP alternative transport URIs.

| Name                | Query | Validity | Description                            | Value      |
|---------------------|-------|----------|----------------------------------------|------------|
| CoAP Transport Type | tt    |          | Transport type requested by the client | xsd:string |

Table 3: The "tt" RD parameter

The "tt" parameter extends the Resource Directory's rd-lookup interface. The "tt" parameter queries existing registrations, and MUST NOT be used with the Resource Directory's registration and update interfaces.

The following example shows a client performing a lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/ep?tt="coap+tcp"

Res: 2.05 Content
</rd/1234>;at="coap+tcp://[2001:db8:f1::2]";ep="node1";ct="40"
```

The following example shows a client performing a resource lookup for endpoints supporting TCP:

```
Req: GET /rd-lookup/res?rt=temperature&tt="coap+tcp"

Res: 2.05 Content
<coap+tcp://[2001:db8:f1::2]/temperature>;ct=0;rt="temperature";
if="core.s";anchor="coap+tcp://[2001:db8:f1::2]"
```

The following example shows a client performing a lookup for endpoints supporting SMS i.e. discovering SMS transports for sleepy nodes and using SMS to communicate with the endpoint:

Req: GET /rd-lookup/ep?et=oic.d.switch&tt="coap+sms"

Res: 2.05 Content

```
</rd/2345>;at="coap+sms://0015105550101/";ep="node5";
 et="oic.d.switch";ct="40",
</rd/4521>;at="coap+sms://0015105550202/";ep="node8";
 et="oic.d.switch";ct="40"
```

## 5. CoAP Alternative-Transport Option

The CoAP Alternative-Transport Option can be used by CoAP clients and CoAP servers in both Request and Response messages in constrained environments where a CoRE Resource Directory is not present.

Figure 4 depicts the properties of the Alternative-Transport Option.

| No. | C | U | N | R | Name                  | Format | Length | Default |
|-----|---|---|---|---|-----------------------|--------|--------|---------|
| 66  |   | x | - | x | Alternative-Transport | string | 0-1034 | (none)  |

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figure 4: The Alternative-Transport Option

When included in a Request message, this option is used by the client in 2 possible ways. In the first case, a CoAP client can include the Option with Length 0 to retrieve all alternative transports from a CoAP server. In response to the client, the server includes base URI for each transport in its own Option. In the second case, a CoAP client can include the Option with a specific value in a CoAP Request, and the CoAP server returns the base URI(s) for the specified transport. If the specified transport by a CoAP client returns multiple results on a CoAP server, the server returns all base URIs of the transport in the response, each base URI in its own Option.

A CoAP client can also use this Option to retrieve several transports at once by including multiple Options in the request to a CoAP server. If any of the specified transports is supported by the

server, the server returns all base URIs in its own option. There can be more than 1 result for any of the transports so that each transport base URI is still included in the response in its own option.

Figure 5 describes a simple interaction between a client and a server, in which the client uses an Alternative-Transports Option with a null value to discover and retrieve all the available transports from the server, as part of a GET operation to retrieve a resource representation. The server responds with a CoAP Response message which contains the resource representation as a payload. In addition, the server also supplies multiple Alternative-Transport Options in the message, with each Option containing the base URI for an available transport. In this case the base URIs returned for TCP-based and WebSocket transports indicate their availability over a non-standard port.

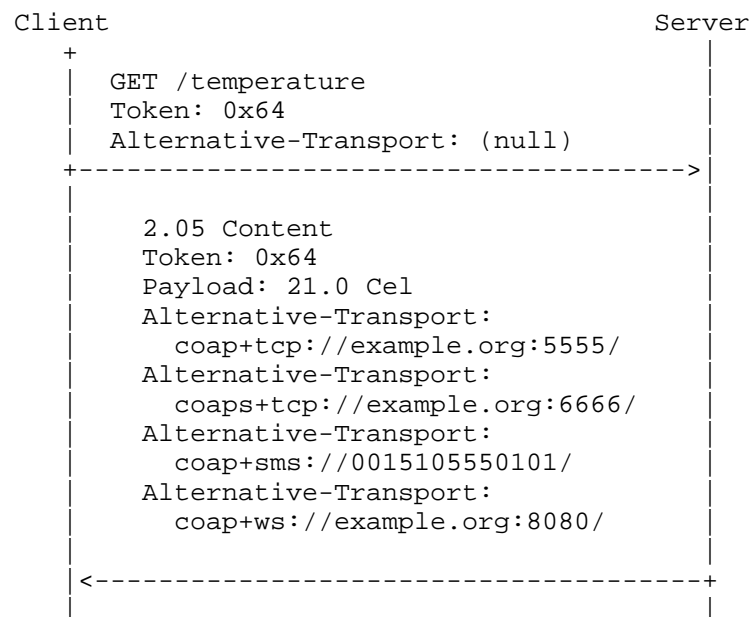


Figure 5: Requesting all available alternative transports on the server, and their locations

Alternatively, a client can also request for the availability of a specific transport on the server, as shown in Figure 6. Here, the CoAP Request contains Alternative-Transport Options with values set to request the Base URIs for TCP-based endpoints.

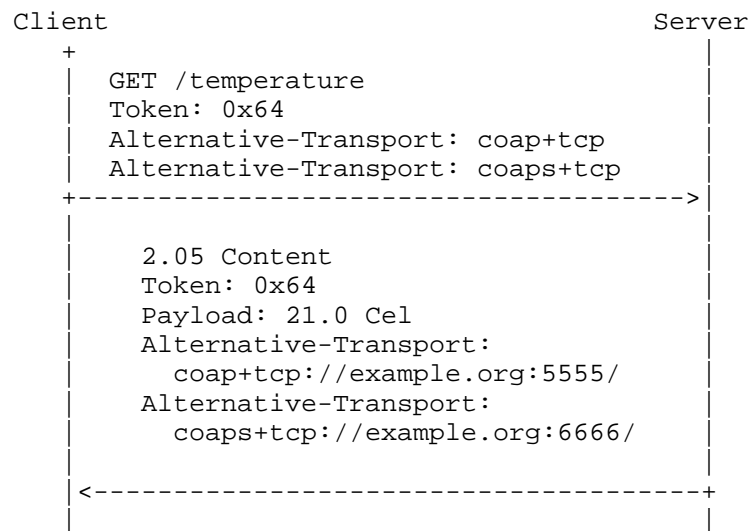


Figure 6: Requesting TCP-based alternative transports on the server, and their locations

A client may also request a subset of available transports on the server, by providing multiple Options, each having a single transport identifier. The server likewise responds to the client request by supplying the requested transport information. This is shown in Figure 7.

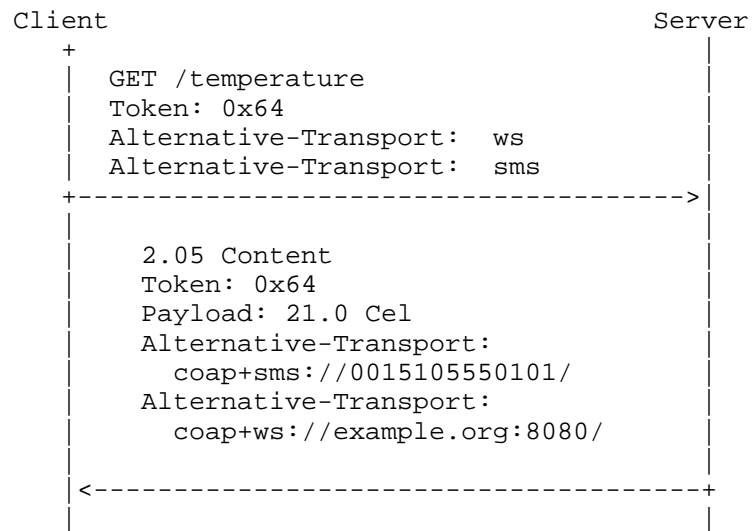


Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

## 6. The 'ol' CoRE Link Attribute

In the majority of cases, it is expected that an origin server would expose all its resources uniformly on its available transports or endpoint addresses. Exceptions can exist however, where alternate locations are made available on a per-resource basis. For such cases, a new 'ol' ("other locations") attribute is provided. One or more 'ol' attributes are used to provide base URIs from which a specific resource can be reached. Allowing per-resource endpoint or transport availability enables specific functions such as firmware updates or hardware-specific operations. It also facilitates mapping to and from OCF-based resource-specific endpoint descriptions. Note that the use of 'ol' is orthogonal to using 'at' as shown in Section 6.2.

### 6.1. Using /.well-known/core

REQ: GET /.well-known/core

RES: 2.05 Content

```
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor",
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
 ol="coap://server2.example.com"
```

## 6.2. Using CoRE Resource Directory

Req: POST coap://rd.example.com/rd  
 ?ep=node1&at=coap+tcp://server.example.com&at=coap+ws://server.example.com:5683/ws/

Content-Format: 40

Payload:

```
</sensors/temp>;ct=41;rt="temperature-f";if="sensor",
</sensors/door>;ct=41;rt="door";if="sensor",
</sensors/light>;if="sensor"; ol="http://[FDFD::123]:61616";
 ol="coap://server2.example.com"
```

Res: 2.01 Created

Location: /rd/4521

## 7. IANA Considerations

This document requests the registration of new RD parameter types "at" and "tt".

The following entry needs to be added to the CoAP Option Numbers Registry:

| Number | Name                   | Reference       |
|--------|------------------------|-----------------|
| 66     | Alternative-Transports | (this document) |

## 8. Security Considerations

When multiple transports, locations and representations are used, some obvious risks are present both at the origin server as well as by requesting clients.

When a client is presented with alternate URIs for retrieving resources, it presents an opportunity for attackers to mount a series of attacks, either by hijacking communication and masquerading as an alternate location or by using a man-in-the-middle attack on TLS-based communication to a server and redirecting traffic to an alternate location. A malicious or compromised server could also be used for reflective denial-of-service attacks on innocent third parties. Moreover, clients may obtain web links to alternate URIs containing weaker security properties than the existing session.

## 9. Acknowledgements

Thanks to Christian Amsuess, Klaus Hartke, Jaime Jimenez and Jim Schaad for comments and reviewing this draft. Teemu Savolainen was involved in initial discussions about protocol negotiations and lifetime values. Zach Shelby provided significant suggestions on how the Resource Directory can be employed and extended in place of link attributes and relation types.

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-resource-directory]  
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-14 (work in progress), July 2018.
- [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>>.

### 10.2. Informative References

- [I-D.silverajan-core-coap-alternative-transport]  
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-11 (work in progress), March 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>>.

[WWWArchv1] <http://www.w3.org/TR/webarch/#uri-aliases>, "Architecture of the World Wide Web, Volume One", December 2004.

## Appendix A. Change Log

### A.1. From -08 to -09

Using "tt" and "Alternative Transports" updated.

### A.2. From -07 to -08

Added example of energy constrained CoAP server

Updated examples of using "at" and "tt"

"at" and "ol" are no longer comma-separated URI lists.

### A.3. From -06 to -07

Added support for 'ol' Link attribute

### A.4. From -05 to -06

Added support for CoAP Alternative-Transports Option

### A.5. From -04 to -05

Freshness update

### A.6. From -03 to -04

Removed previously introduced link attribute and relation types

Initial foray with Resource Directory support

### A.7. From -02 to -03

Added new author

Rewrite of "Introduction" section

Added new Aims Section

Added new Section on Node Types

Introduced "al" Active Lifetime link attribute

Added new Section on Observing transports and resources

Security and IANA considerations sections populated

A.8. From -01 to -02

Freshness update.

A.9. From -00 to -01

Reworked "Introduction" section, added "Rationale", and "Goals" sections.

Authors' Addresses

Bilhanan Silverajan  
Tampere University of Technology  
Korkeakoulunkatu 10  
FI-33720 Tampere  
Finland

Email: bilhanan.silverajan@tut.fi

Mert Ocak  
Ericsson  
Hirsalantie 11  
02420 Jorvas  
Finland

Email: mert.ocak@ericsson.com

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: July 28, 2018

M. Veillette, Ed.  
Trilliant Networks Inc.  
January 24, 2018

Constrained YANG Module Library  
draft-veillette-core-yang-library-02

Abstract

This document describes a YANG library that provides information about all the YANG modules used by a constrained network management server (e.g., a CoAP Management Interface (CoMI) server). Simple caching mechanisms are provided to allow clients to minimize retrieval of this 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 <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 28, 2018.

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. Major differences between ietf-constrained-yang-library<br>and ietf-yang-library . . . . . | 3  |
| 2. Terminology and Notation . . . . .                                                           | 3  |
| 3. Overview . . . . .                                                                           | 4  |
| 3.1. Tree diagram . . . . .                                                                     | 4  |
| 3.2. Description . . . . .                                                                      | 5  |
| 3.2.1. modules-state . . . . .                                                                  | 5  |
| 3.2.2. modules-state/hash . . . . .                                                             | 5  |
| 3.2.3. modules-state/module . . . . .                                                           | 5  |
| 4. YANG Module "ietf-constrained-yang-library" . . . . .                                        | 5  |
| 5. IANA Considerations . . . . .                                                                | 10 |
| 5.1. YANG Module Registry . . . . .                                                             | 10 |
| 6. Security Considerations . . . . .                                                            | 10 |
| 7. Acknowledgments . . . . .                                                                    | 11 |
| 8. References . . . . .                                                                         | 11 |
| 8.1. Normative References . . . . .                                                             | 11 |
| 8.2. Informative References . . . . .                                                           | 11 |
| Author's Address . . . . .                                                                      | 12 |

## 1. Introduction

WARNING: Both this contribution and the CoMI protocol [I-D.ietf-core-comi] need to be reviewed to verify their compatibility with the "Network Management Datastore Architecture" (NMDA). See [I-D.dsdt-nmda-guidelines], [I-D.ietf-netconf-rfc7895bis], [I-D.ietf-netmod-revised-datastores] and [I-D.ietf-netconf-nmda-restconf] for more details.

The YANG library specified in this document is available to clients of a given server to discover the YANG modules supported by this constrained network management server. A CoMI server provides a link to this library in the /mod.uri resource. The following YANG module information is provided to client applications to fully utilize the YANG data modeling language:

- o module list: The list of YANG modules implemented by a server, each module is identified by its assigned YANG Schema Item Identifier (SID) and revision.
- o submodule list: The list of YANG submodules included by each module, each submodule is identified by its assigned SID and revision.
- o feature list: The list of features supported by the server, each feature is identified by its assigned SID.

- o deviation list: The list of YANG modules used for deviation statements associated with each YANG module, each module is identified by its assigned SID and revision.

#### 1.1. Major differences between ietf-constrained-yang-library and ietf-yang-library

YANG module 'ietf-constrained-yang-library' targets the same functionality and shares the same approach as YANG module ietf-yang-library. The following changes with respect to ietf-yang-library are specified to make ietf-constrained-yang-library compatible with SID [I-D.ietf-core-yang-cbor] used by CoMI [I-D.ietf-core-comi] and to improve its applicability to constrained devices and networks.

- o YANG module 'ietf-constrained-yang-library' extends the caching mechanism supported by 'ietf-yang-library' to multiple servers of the same type. This is accomplished by replacing 'module-set-id' by a hash of the library content.
- o Modules, sub-modules, deviations and features are identified using a numerical value (SID) instead of a string (yang-identifier).
- o The "namespace" leaf, not required for SIDs, but mandatory in 'ietf-yang-library' is not included in 'ietf-constrained-yang-library'.
- o Schemas can be located using the already available module or sub-module identifier (SID) and revision. For this reason, support of module and sub-module schema URIs have been removed.
- o To minimize their size, each revision date is encoded in binary.

## 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 module
- o submodule
- o feature
- o deviation

The following terms are defined in [I-D.ietf-core-yang-cbor]:

- o YANG Schema Item iDentifier (SID)

The following terms are defined in [I-D.ietf-core-comi]:

- o client
- o server

The following terms are used within this document:

- o library: a collection of YANG modules used by a server.

### 3. Overview

The "ietf-constrained-yang-library" module provides information about the YANG library used by a given server. This module is defined using YANG version 1 as defined by [RFC7950], but it supports the description of YANG modules written in any revision of YANG.

#### 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 [I-D.ietf-netmod-yang-tree-diagrams].

```
module: ietf-constrained-yang-library
 +--ro modules-state
 +--ro hash binary
 +--ro module* [sid revision]
 +--ro sid comi:sid
 +--ro revision revision
 +--ro feature* comi:sid
 +--ro deviation* [sid revision]
 | +--ro sid comi:sid
 | +--ro revision revision
 +--ro conformance-type enumeration
 +--ro submodule* [sid revision]
 +--ro sid comi:sid
 +--ro revision revision

notifications:
 +---n yang-library-change
 +--ro hash -> /modules-state/hash
```

### 3.2. Description

#### 3.2.1. modules-state

This mandatory container specifies the module set identifier and the list of modules supported by the server.

#### 3.2.2. modules-state/hash

This mandatory leaf contains the hash of the library content. The value of this leaf MUST change whenever the set of modules and submodules in the library changes. This leaf allows a client to fetch the module list once, cache it, and only re-fetch it if the value of this leaf has been changed.

If the value of this leaf changes, the server also generates a 'yang-library-change' notification.

#### 3.2.3. modules-state/module

This mandatory list contains one entry for each YANG module supported by the server. There MUST be an entry in this list for each revision of each YANG module that is used by the server. It is possible for multiple revisions of the same module to be imported, in addition to an entry for the revision that is implemented by the server.

### 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 {
 namespace
 "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
 prefix "lib";

 import ietf-comi {
 prefix comi;
 }

 organization
 "IETF CORE (Constrained RESTful Environments) 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 contains the list of YANG modules and submodules implemented by a server.

Copyright (c) 2016 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.: replace XXXX with actual RFC number and remove  
// this note.

// RFC Ed.: update the date below with the date of the RFC  
// publication and remove this note.

```
revision 2018-01-20 {
 description
 "Initial revision.";
 reference
 "RFC XXXX: Constrained YANG Module Library.";
}
```

```
/*
 * Typedefs
 */
```

```
typedef revision {
 type binary {
 length "4";
 }
 description
```

```
 "Revision date encoded as a binary string as follow:
 - First byte = Century
 - Second byte = Year (0 to 99)
 - Third byte = Month (1 = January to 12 = December)
 - Forth byte = Day (1 to 31)";
}

/*
 * Groupings
 */

grouping identification-info {
 description
 "YANG modules and submodules identification information.";

 leaf sid {
 type comi:sid;
 mandatory true;
 description
 "SID assigned to this module or submodule.";
 }

 leaf revision {
 type revision;
 description
 "Revision date assigned to this module or submodule.
 A zero-length binary string is used if no revision
 statement is present in the YANG module or submodule.";
 }
}

identity module-set {
 description
 "Base identity from which shared module-set identifiers
 are derived.";
}

/*
 * Operational state data nodes
 */

container modules-state {
 config false;
 description
 "Contains information about the different data models
 implemented by the server.";

 leaf hash {
```

```
 type binary {
 length "8..32";
 }
 mandatory true;
 description
 "A server-generated hash of the contents of the library.
 The server MUST change the value of this leaf each time
 the content of the library has changed. The hash function
 and size are not specified, but shall be collision
 resistant."
 }

list module {
 key "sid revision";
 description
 "Each entry represents one revision of one module
 currently supported by the server.";

 uses identification-info;

 leaf-list feature {
 type comi:sid;
 description
 "List of YANG features from this module that are
 supported by the server, regardless whether
 they are defined in the module or in any
 included submodules."
 }

 list deviation {
 key "sid revision";
 description
 "List of 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.

 Deviation modules MUST also be present in the 'module'
 list, with the same sid and revision values and the
 'conformance-type' set to 'implement'."

 uses identification-info;
 }

 leaf conformance-type {
 type enumeration {
 enum implement {
```

```
value 0;
description
 "Indicates that the server implements one or more
 protocol-accessible objects defined in the YANG
 module identified in this entry. This includes
 deviation statements defined in the module.

 For YANG version 1.1 modules, there is at most one
 module entry with conformance type 'implement' for
 a particular module, since YANG 1.1 requires that
 at most one revision of a module is implemented.

 For YANG version 1 modules, there SHOULD NOT be more
 than one module entry for a particular module.";
}
enum import {
 value 1;
 description
 "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 module entries for the same module MAY
 exist. This can occur if multiple modules import
 the same module, but specify different revision-dates
 in the import statements.";
}
}
mandatory true;
description
 "Indicates the type of conformance the server is claiming
 for the YANG module identified by this entry.";
}

list submodule {
 key "sid revision";
 description
 "Each entry represents one submodule within the
 parent module.";
 uses identification-info;
}
}
}

/*
 * Notifications
 */
```

```
notification yang-library-change {
 description
 "Generated when the set of modules and submodules supported
 by the server has changed.";

 leaf hash {
 type leafref {
 path "/lib:modules-state/lib:hash";
 }
 mandatory true;
 description
 "New hash value.";
 }
}
}
<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` [RFC7895], 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-comi]  
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-02 (work in progress), December 2017.
- [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-05 (work in progress), August 2017.
- [I-D.ietf-netmod-yang-tree-diagrams]  
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-05 (work in progress), January 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>>.
- [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>>.

### 8.2. Informative References

[I-D.dsdt-nmda-guidelines]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Guidelines for YANG Module Authors (NMDA)", draft-dsdt-nmda-guidelines-01 (work in progress), May 2017.

[I-D.ietf-netconf-nmda-restconf]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", draft-ietf-netconf-nmda-restconf-02 (work in progress), January 2018.

[I-D.ietf-netconf-rfc7895bis]

Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-03 (work in progress), January 2018.

[I-D.ietf-netmod-revised-datastores]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-10 (work in progress), January 2018.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<https://www.rfc-editor.org/info/rfc7895>>.

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)